SIMULATION OF A TANK BATTLE ON COMPUTERS

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

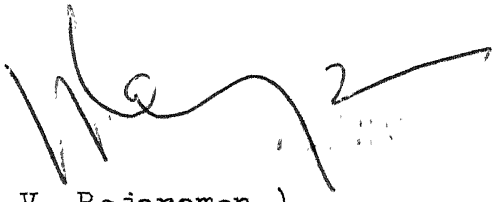
By

Capt. VINAY SAGAR

to the
DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR
SEPTEMBER, 1981

CERTIFICATE

This is to certify that the thesis entitled 'SIMULATION OF A TANK BATTLE EXERCISE ON COMPUTERS' by Capt. Vinay Sagar is a record of work carried out under my guidance and has not been submitted elsewhere for a degree.



Sept. 1981

(V. Rajaraman)
Professor
Electrical Engineering Department
Indian Institute of Technology
KANPUR

ACKNOWLEDGEMENT

I would like to thank Dr. V. Rajaraman, my thesis supervisor for all the help and guidance he has given me. It has given me great pleasure to work under him and get his valuable advice.

I would also like to thank Mr. J.S. Rawat for typing out this thesis neatly.

Capt. Vinay Sagar

ABSTRACT

An interactive simulator for tactical exercise with battle tanks has been described here. The effort has been to make this simulator a logical extension of a sand model exercise, so that a more realistic exercise which is closer to real exercises in field may be achieved.

The topographical details of a map area required to be used, is stored in the memory. The user is permitted to give commands to MOVE and FIRE. The move and fire is simulated taking into consideration the topographical details, characteristics of armour and the firing capability of tanks. The REPORT facility in the simulator helps the user in gaining information regarding tank status and other details used for taking decisions.

It is considered that simulator when fully developed will form a useful aid in exercising a squadron as a team, in which the squadron commander along with his troop commanders can participate.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

In Army, tactical exercises form an important part in troop training. These exercises involve the process of making decisions under visualised situation to resemble true combat situation. Troop commanders are exercised under constraints of carefully planned rules, as they would under combat conditions.

Field Exercises are very costly and have other effects like cost in fuel consumption, wear and tear of equipment and wastage. Wear and tear of tanks is an important factor which reduces the tank life considerably. As such frequent field exercises involving tanks are not possible. Moreover the area available for such exercises is restricted. Built up areas cannot be used for exercise and firing because of the damage it would cause. Also exercise cannot be done in critical areas like borders etc. Therefore at present the use of 'Sand Model' to exercise the personnel in tactics is the most commonly used method. Any form of exercise being at best, a poor substitute for experience, the emphasis is therefore more on realism and greater detail. The objective of this thesis is to develop

a computer based interactive simulation in an effort to replace the sand model exercise with increased reality and more details.

1.1 The Importance of a Simulator:

The simulator comes in between the sand model and field exercise and can have certain advantages over it. Some of the essential features of the simulator are described in the succeeding paragraphs.

The preparation of a sand model is time consuming and even then all the details cannot be represented on it. In constrast in the computer simulation the representation of the topography exists in the memory of the computer which can be more exhaustive than a sand model.

The hypothetical vehicle i.e. tank in this case is 'moved' and can 'fire' taking into account the factors that govern movement such as terrain, gradient, mobility and for fire accuracy of gun,rate of fire and penetration etc.

Visibility and sighting of the tanks is decided by the computer depending on the topographical information available with the computer. Factors that decide visibility like cover,concealment and line of sight is accounted for.

Communication, queries and replies between the simulated tanks by the commanders can be done through the computer.

Freedom of action is available to the participating personnel and pre-determined action is curtailed so that the exercise is allowed to take its course of action.

1.2 Review of Similar work done earlier:

In recent past a lot of thought is going in this direction of simulation on computer mainly because of the economics of time, cost and effort saving that it offers.

Known similar work has been done on 'war gaming' [1], in which a game is described where in two opposing forces are made to 'move' from one point to another according to a battle plan. During this course of movement analysis is done so that if enemy is sighted then it fires a fixed number of rounds/shells on it. In the end the success or failure is evaluated in terms of quantum of forces that have reached their destination.

Another work that has been done has been on 'Computers and Tactical war gaming' [2]. This is a theoretical description on how simulation can be done for a tank battle at squadron level. This work describes how computer can be

used to exercise the commanders i.e. squadron and troop commanders in taking decision in near real life situation. This is more of an interactive process as would be seen in an exercise. The idea for this thesis is based on this work and simulation programmes have been developed in PASCAL for its implementation on Dec 1090.

1.3 Goal of this Thesis:

The goal of this thesis is 'To write the necessary programmes to realise the command and communication reportoire in a higher level computer language for simulation of tank battle'.

To achieve the above goal the following essential features are considered necessary:

a) A detailed description of battle area topography to be stored in the memory. Roads, rivers, obstacles, minefields, untankable country etc. are depicted so that the exercise can be simulated on any desired type of terrain or area for which map is available.

b) Detailed description of every participating element, including its characteristics of tank and armament.

c) The tanks in the memory are 'moved' taking into account the factors that govern movement like terrain, obstacles etc. Gains and losses are decided upon, appropriate to firepower, accuracies of the gun, rate of fire, penetration etc.

d) Factors deciding visibility such as cover, concealment and direct visibility are catered for.

e) Tactical decisions are made by the users. They type in command and querries and receive replies in a normal form.

f) A set of factors additional to that mentioned above like artillery, anti-tank weapons,air support could be included to give more realistic and complex view.

To complete the simulation of a squadron level tank exercise, in addition to this part, development of the requisite interface to couple the necessary consoles and other aids used is required. This could be done as a further development to this thesis. When this is done, then this could be used to exercise the squadrons i.e. the squadron commander along with troop commanders for a tank battle.

1.4 Chapter Summaries:

In the next chapter we review the computer aids available for tactical exercises and war gaming. In this methods available for war gaming, the importance of Tactics/Strategy and any known work in this field will be discussed.

The complete 'over view' of the system will be given in Chapter III. This will give the block diagram of the system developed and what is expected of each block. Interconnections between these blocks and how to use the developed system will be described.

Chapter IV will give some specific algorithms that have been used. Approximations and assumptions made in developing this system will be discussed with possible improvements and the type of computer required for implementation of such a programme.

The graphic aid used by the simulator and a 'test exercise' will be described in the next chapter. The last chapter gives the conclusions and suggestions for further work.

CHAPTER II

WAR GAMING ON COMPUTER

War gaming has become a recognised technique in relatively recent times,but its essense has been a part of the pre-planning that military commanders have done through the ages. The military services would define a war game as a simulation, by whatever means, of a military operation involving two or more opposing forces, conducted, using rules, data and procedures designed to depict an actual or assumed real life situation. The simulated warfare provides a means of gaining experience, identifying errors or short-comings and improving skills without paying the penalties of the real war. Any thing in this world which is as costly as a war impels a search for its substitute.

2.1 War Gaming:

In military most of the important problems are amenable to analytical study. At present war gaming is mainly used for the following purposes:

a) To train military personnel

b) To test plans of commanders

c) For research, to explore new concepts.

Generally the above three purposes are clearly specified and well defined for each game. In some cases it could be used for multiple purpose as well, like for training as well as testing of plans. Games for research are usually conducted to study matters associated with new concepts. They also include the evaluation of plan and/or the formulation of doctrine in the application of new concepts. These three major purposes of war gaming are very much inter-related and also represent the stages of progressive development of war gaming.

War games normally attempt to develop and use models to represent the actions in a game. The model is important in a simulation. In a war game some factors not explicitly quantified in the model can be added by the controller, specified in the rules as input data, or covered as rules in the game assumptions. So effective models of battle can be built for some of the important factors involved in combat, but not all. Too little is known about the geometry of battle. Even information on the rate of casualty production and the resulting effects is inadequate to build precise mathematical equations. Formulae and model from historical records have some validity but have predictive value in a general way only. Although such models are

useful, they do not provide all the interactions and measures of effect needed for many analytical purposes. The object of the game will then be served by comparative rather than absolute casualty figures. Such a model is only as good as the capability of the model builder to build military sophistication and battle logic-expertise into the model.

2.2 Tactics/Strategy:

Some basic characteristics of war games normally found in all of them are:

a) Every war games simulates a military operation (irrespective of phase or manner of gaming).

b) Each game involves two or more opposing forces.

c) Each war game is conducted in accordance with data, rules and procedures acceptable to the military profession.

d) Every war game represents an actual or assumed real life situation.

An activity cannot be termed a war game unless military forces are involved in movements or operations accompanied by the clash of arms or the threat of such a clash. War

games are played only under conditions of simulated war-fare i.e. shooting, attack, or invasion by one force upon the territory of another. War game rarely consider non-shooting situations unless the situation is an interlude during which forces get into position or range to fight.

Therefore a war game is built upon a situation such as exists or realistically could exist in some locale under certain assumed conditions. The situation is described in detail sufficient for the commander to visualise the conditions under which he must conduct military operation.

Taking the above conditions in view the following ingredients are necessary for war gaming:

a) Knowledgeable personnel
b) Special facilities and equipments
c) Standard data, rules and procedures
d) A description of the situation to be represented in the game.

2.3 Other work in this Area:

The 'Tin Soldier' which became operable in 1952, is believed to be the first war game designed as a mathematical model for use on a computer in analytical research studies.

This war game is simple and is described as follows. A map is given for the area of interest and on it are

marked the various status of troops of pre-battle conditions for both the opposing forces. It essentially involves two forces, own and enemy forces. Then one man each is permitted to 'head' these forces. Their aim being to guide their forces in an imaginary battle. The battle is then organised to progress under the rules agreed as follows.

a) Each man is permitted to move his tanks/forces that would correspond to 't' seconds elapsed time on real terrain, consistent with the mobility capabilities of the tank.

b) After each man moves his tanks/troops according to his military good sense, tanks within range are assumed to bring fire on the opposition. The winner of tank duel is decided by flipping a loaded coin that is supposed to express the odd on the battle outcome in actual fighting between these tanks.

c) The battle is over as soon as all tanks on either side have been knocked out or have made successful withdrawal from the battle field.

The next step to this was the computerised Monte Carlo version called 'The maximum Complexity Computer Battle', capable of dealing with more of the variables of actual battle. The technique employed is called 'pseudo-random

number generator'. Every time this random number generator is referred, a large sequence of numbers (R) are computed, so that the sequence is practically random as far as the observer is concerned. If the probability of an event 'E' taking place is 'p', p being between '0' to '1', the following procedure simulates the occurrance of E:

if $R < p$, E occurs otherwise not.

Every time procedure is called, R is re-computed and compared with 'p'. Over a large number of calls, 'E' will occur the specified percentage of times. In addition in deciding the occurrence or non-occurrence of probabilistic events, the Monte-Carlo technique also provides for the determination of variables used in simulation in a probabilistic manner.

While the technique described here is satisfactory as a first approximation, it is necessary to add several refinements in practice. The uniform distribution of random variables is not common in practice.

Other methods used for formulation are the 'Next-Event formulation' and 'Fixed time step formulation'. The basic difference between the two is of time of computation. In the first i.e. next event formulation the time period used

is the time that will elapse between two subsequent events. Hence this time period varies as the simulation progresses. Whereas in 'fixed time', the time period of computation is kept constant to a pre-determined value.

With the 'next event' model, the processing of event occurences proceed occurence by occurence as follows. The simulation time is advanced to the time of the earliest future occurence. That occurence is processed, then the simulation time is advanced to the time of the (new) earliest occurence and that occurence is processed and so on. However, with the fixed time-step model, the processing of event occurences does not proceed occurence by occurence in this way. Rather it proceeds by group or batches of event occurences. All event occurences during the time step are processed together.

With the 'next-event' model the need to choose an arbitrary and artificial increment does not exist, and the pit-fall of picking a suitable increment is avoided entirely. In this respect, the next event formulation is preferable to the fixed time step formulations. Further more in a next-event model two event occurences are not processed as simultaneous events unless they bear identical occurence times. In a time-step model, on the other hand, batches of

non-simultaneous occurences are grouped together and made to appear simultaneous. Thus, simultaneity is created where it need not exist. In order to process simultaneous occurences,any simulation model must make use of some rule for determining the sequence in which to process the occurences. Fixed time-step simulation then, often takes a group of non-simultaneous event occurences, forces them to be simultaneous and process them in some sequence.

One situation, however, may require a user to use a time-step model instead of a next-event model. This is the situation where a next-event model takes a prohibitively long time to run on a computer and where the artificialities and inaccuracies of the time step model do not destroy the usefulness of the results. If the time increment, in fixed time step is sufficiently large, it is possible that the fixed time-step simulation may progress significantly faster than the corresponding next event simulation.

In this thesis the model of fixed time-step which can be modified to different values during the simulation is used. The capacity to modify the time step gives the option to the user to make a trade off between accuracy and speed. Thus the user is able to decide during the

simulation process, depending on the status or the situation that exists at that moment. If the user feels that the time step is to be reduced or increased, he can do so at that time.

2.4 Inter-active Simulation:

On computer there are two main types of simulation, one is complete computer simulation and another is man-computer simulation or interactive simulation. In the all computer simulation the input data and the rules are pre-described and given to the computer. The computer then simulates that action and reaches the result or conclusion. In this a number of runs would yield similar results because of the uniform strategy/decision of processing. Whereas inter-active simulation is different and would give different results with each user.

In this ,instead of producing a complete history automatically in one continuous operation such a routine processes events until an event is reached where a human decision is necessary or additional input is required. Then the routine causes the computer to wait for appropriate action to be taken before it starts another 'computer phase' of simulation. This is near to real time exercises

conducted by the armed forces and therefore better suited to simulate battle exercises. The simulator described here uses this interactive process of simulation.

Each interval during which the computer waits for input or for decisions is called a 'wait phase' of simulation. During this phase one or more team is in action. A team may be deciding on the movements of units, determining firing and communication policies to be followed, assessing information they have received about their own and enemy's units, and so on. The end of this phase occurs when the information requested by the computer is made available to restart the computer simulation routine.

The most obvious difference between a computer routine for man-computer gaming and a complete computer simulation routine is that the former must be able to 'converse' with the human players. For instance, the routine must be able to inform the players whenever any unit require repair, fuel etc. The routine must also inform the human players of enemy detections, firing of weapons, and damage sustained or inflicted. Each message is delivered through a computer output unit to the appropriate team or player.

An interactive game has some undesirable features which are associated with the alternating of the game. For example,

a 'wait phase' once begun must continue until it produces the decisions or other information which allow another computer phase to be started. Since this corresponds to stopping battles while decisions are being made, it is a poor representation of a real situation. Any attempt to place time limits on the 'wait phase' of the game requires rules which bear little relation to the real situation that one is trying to portray. Also the computer phase of the game must stop whenever any team must furnish information required by the computer simulation routine. This may cause delays unexplainable to one team or the other. For such reasons as these, use of 'synchronized' man-computer game is used where both 'wait' and 'computer' phases run continuously and concurrently with the use of 'clock'.

CHAPTER III

OVER VIEW

Tank battle simulation consists of essentially the storage of data concerning each tank and its subsequent updating depending on the commands given by the user. The data on each tank is stored in a record TNKDAT and has the following fields.

TYPNO: (1..5); Gives the type of Tank. Caters for a maximum of 5 types.

MAPCORD: X,Y; (INTEGER); Gives location of the tank in X and Y co-ordinates.

ORNTASN: (0..360); Gives the direction in which the tank is facing.

AMNCAP :TYP1, TYP2, TYP3: (INTEGER); Gives the amount of ammunition that is available with the tank. Caters for 3 different type of ammunition.

FUELCAP: (INTEGER); Gives the Quantity of fuel that is available with tank.

ALT : (INTEGER); Altitude in metres where the tank is located.

LODTME: (INTEGER); The time taken to load the shells before it is fired once the command to fire has been given.

STATUS:

BOGGED:(BOOLEAN); States whether the tank is bogged.

MOVING: (BOOLEAN); Whether the tank is moving, used to calculate rate of fire etc.

CANMOV:(BOOLEAN); Used when command to move is given.

FIRING:(BOOLEAN);States whether the tank is firing on target.

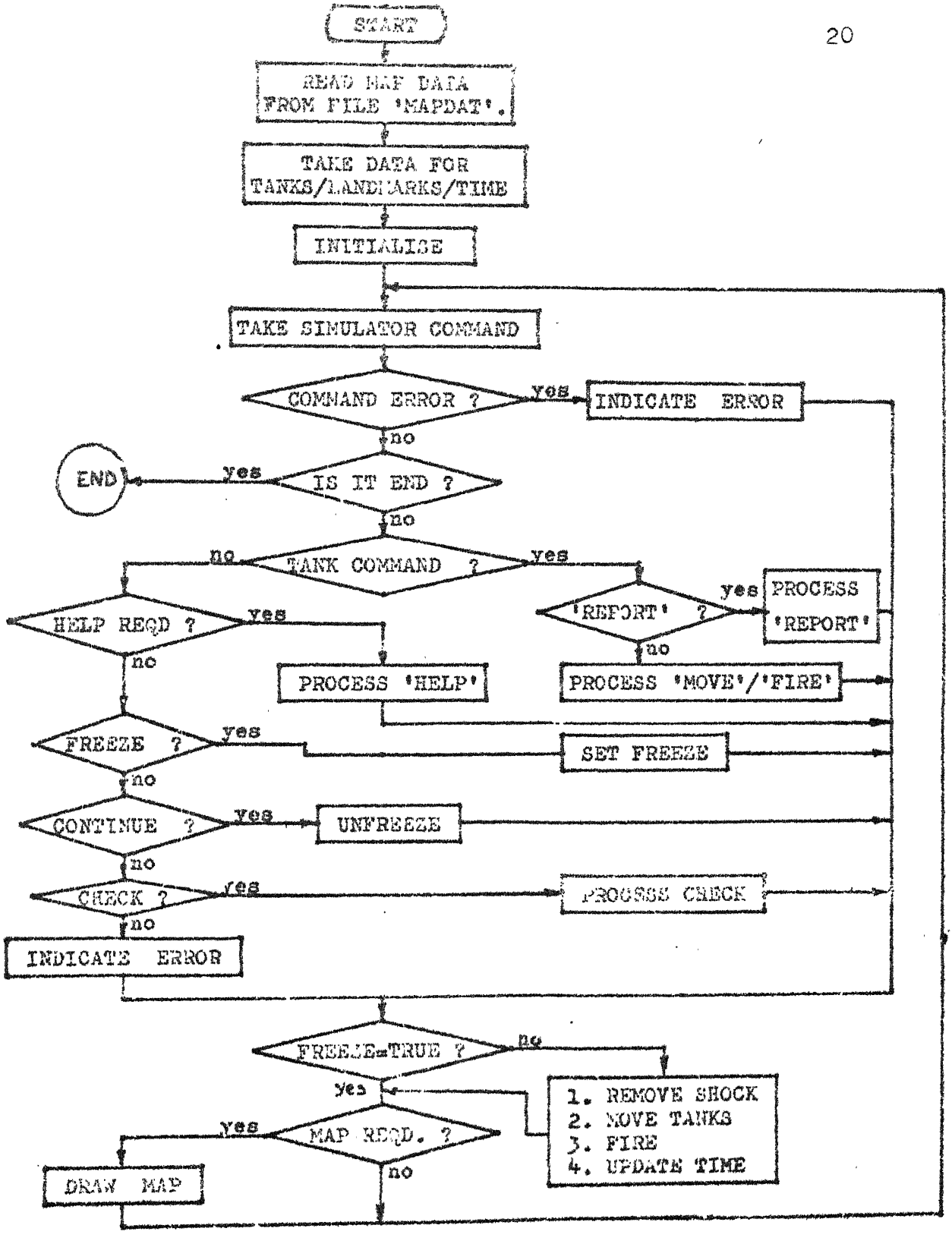
CUPOLACLOSED:(BOOLEAN); Used for visibility calculation.

NUETRALISED:(BOOLEAN); Whether tank is incapacitated and can take no further part.

SHOCKED:(BOOLEAN); Temporarily out of action, will not respond in that duration.

Various data from these fields can be 'read' through REPORT procedures which help the user in taking decision. These data also form the base for generation of other tables.

The simulation of MOVE and FIRE are done through the use of temporary tables which are processed to change the records of the affected tank in TNKDAT.


START
READ MAP DATA FROM FILE 'MAPDAT'.
TAKE DATA FOR TANKS/LANDMARKS/TIME
INITIALISE
TAKE SIMULATOR COMMAND
COMMAND ERROR ?
yes
INDICATE ERROR
no
END
yes
IS IT END ?
no
no
TANK COMMAND ?
yes
HELP REQD ?
yes
no
'REPORT' ?
yes
PROCESS 'REPORT'
no
PROCESS 'HELP'
PROCESS 'MOVE'/'FIRE'
FREEZE ?
yes
SET FREEZE
no
CONTINUE ?
yes
UNFREEZE
no
CHECK ?
yes
PROCESS CHECK
no
INDICATE ERROR
FREEZE=TRUE ?
no
yes
1. REMOVE SHOCK
2. MOVE TANKS
3. FIRE
4. UPDATE TIME
yes
MAP REQD. ?
no
DRAW MAP

The flow chart for the simulation process is shown as Fig. 1. This could be divided into the following sub-systems.

a) Map Generation

b) Initialization

c) Command Processing

d) Report

e) Information/Help

f) Update Records

g) Action

h) Graphic Display

3.1 Goals for Each Block:

Map Generation:

This part deals with the description of map details. This is a time consuming process and hence is separated from the main programme. The map is drawn with an interactive dialogue. The validity of the input with respect to the restrictions imposed for drawing the map are checked and then a file is created for the use of the main programme. This consists of the following 2 parts.

(a) Checking of Input:

Due to the limitations and economy of computer usage

some restrictions have been imposed for describing the details of a map. They are

i) Grid points are limited to a maximum of 40x40.

ii) Each grid point is at a distance of 500 mtr apart.

iii) Details such as rivers, canals etc. have been approximated as straight lines between turning points.

(b) Creating Data Files:

The above facts are checked as and when the map is being drawn. If it is within these constraints then this data is written into a file 'MAPDAT' which forms an input to the main programme. In addition another file 'MAP' is created to facilitate the drawing of map on Graphic terminal and avoid duplicating of calculation.

Initialise:

This is used for starting the simulation of the exercise. This consists of initialising the following:

(a) Map Details:

Reading the map data from file MAPDAT and storing the map details.

(b) Tank Details:

Initialising/reading the characteristics of the type of tanks used and their location at start.

(c) Graphics:

Initialising the graphic device to be used and resetting its data file MAP for its use.

(d) Land marks:

Taking in interactively the various landmarks that are likely to be used during the exercise and storing them for reference in LMTAB.

(e) Time:

If required, updating the 'Time step' to be used for each iteration. Also takes in the start time of the exercise.

(f) Variables:

Initializing the various pointers arrays and variables used in the programme.

Command Processing:

This is the main block of the system. It takes in one command at a time, analysis it and then appropriately calls the block required. The following facilities are available:

a) The command is analysed for its syntax error and if error exists, then this is indicated to the user.

b) Facility to 'freeze' all action and 'un-freeze' it when required.

c) Updates the time that has elapsed since the start time, if action is not frozen.

d) Process 'help' for MOVE, FIRE and REPORT.

e) Change time step if required, during execution of the programme.

Report:

This block is essentially a report back answer to querries addressed to the various tanks. Most of the details are taken from record TNKDAT. The report could be concerning the following.

(a) Location :

This reports the 4 figure grid reference on the map. This could be of

i) Self

ii) Another tank.

(b) Status:

This concerns the various activities that a tank could do or is doing. This refers to whether the tank is

i) Bogged
ii) Moving
iii) Can move
iv) Can shoot
v) Nuetralized

Once a tank is nuetralized then it can take no further part in the exercise and will not respond to any command.

(c) Visibility:

Checks if the other tank or location is visible to the addressed tank taking into consideration the following.

i) Time of day
ii) Equipment available for visibility at that time of day.
iii) Cover available to target
iv) Intermediate obstruction in the line of sight of these 2 points.

(d) Balance:

Balance of 2 commodities can be asked for

i) Fuel: This includes reserve as well

ii) Ammunition: This can be asked for a specific type of ammunition i.e.

(aa) HE (High Explosive)

(ab) AP (Armour Piercing)

(ac) SMOKE

By default, if no particular type is specified, then all types will be listed with their balance.

Information/Help:

This contains the information regarding the various modules, facilities that are available and how to use them. It essentially contains information regarding the following:

(a) HELP IN

i) Commands

ii) Report

iii) Move

iv) Fire

(b) CHECKING

i) List of land marks defined

ii) List of Tanks on move

iii) To change time step

iv) Set/unset drawing of a map

v) Display map.

This can be called for by HELP/CHECK and then choose the information in the type required by you. For 'help' it lists out the possible commands in that field with options. This forms the command communication language and is approximated to the actual commands that are normally used, keeping in mind its application on the computer. Commands available are given as Appendix A.

UPDATE RECORDS

When ever any 'Move' or 'Fire' command is given, details regarding the affected tanks is stored in records for further processing in the 'Action' part. This part does the following.

(a) Checks if the command is correct otherwise the error is indicated to the user.

(b) If the command is correct then it extracts information to be stored in the respective Move/Fire tables. The details are:

i) For Move: This data is stored in a chained record-MOVTAB with following details:

(aa) Starting location

(ab) Destination

(ac) Route - maximum of 10 intermediate points are catered for.

(ad) Current location

(ae) Name of tank

ii) For FIRE: The data for this is stored in record-FIRETAB . It consists of the following details

(aa) Name of tank

(ab) Target location

(ac) Target tank No. if any

(ad) No. of shells to be fired.

(ae) Rate of fire

(af) Hit probability

(ag) Type of shells to be fired

These tables form the basis for the other blocks which do the 'Action' part. Once the action is complete then that particular record is deleted from the chain. Therefore only the currently required information is stored in these chains.

Action:

This is essentially simulating the 'move' and 'fire' action in the system. The input to this is taken from the records FIRETAB for 'Fire' and MOVTAB for 'move'. For every cycle of time the following action part is done.

(a) For MOVE: For each of its record in MOVTAB it does the following:

i) Checks if the status is not nuetralized, if it is not then it will start the move process.

ii) It will then take its next objective and calculate the next grid point that it has to move.

iii) Before 'moving' to the next grid point, checks for its topography whether there is a river, canal, minefield etc. and takes appropriate action.

iv) If it can move to the next grid point then calculates the following

(aa) Time taken for this move

(ab) Fuel spent

and up dates both these quantities.

v) Thus it will continue till either it reaches its destination, is stopped or the time step is over.

(b) Fire: When 'fire' is to be done the following is done for each record of its table 'FIRETAB'.

If the status of the tank is not nuetralized then each time period, it will fire the shells depending on the following

i) Rate of fire of the tank depending on whether it is moving or stationary. This is taken from the tank status in TNKDAT.

ii) Calculate the location of hit taking into consideration the probability of hit and ranging.

iii) Analysing the hit with respect to the target tank i.e. angle of hit, distance and type of ammunition used. Then accordingly changes the status of the effected tank if needed.

iv) It will continue this till the required number of shells are fired or time step finishes or is commanded to stop firing.

Graphic Display:

Then lastly the complete activity is displayed on the Graphic terminal. The display consists of the following

(a) Details of the map which include the following

i) Rivers

ii) Roads

iii) Canals

iv) Minefields

v) Marshy ground

vi) Fields/High grass

(b) The map displays the area of 20x20 km divided into 40x40 grid points i.e. each grid point is at a distance of 500 mtr apart.

(c) For referencing on the map, grid marks are available at 2.5 kms distance (5 grid points).

(d) The 'land marks' defined by the user is also displayed at the location with the assigned name.

(e) Location of all tanks, as they are located at the time of display of the map.

(f) Location of hit of the shell at the time of display if there is any.

(g) Contours are displayed at an interval of 50 mtr. This is done with relative height, with the surrounding ground taken as '0'.

3.2 Interconnections:

The common record for the complete programme is TNKDAT. There are other subsidary records to this record which are used to update this during the exercise.

The map data generated is primarily used for the 'move' and display purposes. This data is kept in GRIDCHAR. When the move is simulated then details regarding the ground is used for calculation of speed, fuel spent and obstacles enroute.

After calculation of fuel spent and new location, they are updated in the record TNKDAT. If there is any change in status like the tank is bogged, nuetralized etc.

the same is changed in the 'status' field of TNKDAT. Therefore the latest information is stored in the record TNKDAT.

When 'Fire' has to be done. The shells are fired from the tank depending on the command given and the same number as and when fired are reduced from its record of that particular tank. The location of hit is calculated and then the effect of shell analysed depending on the type of shell and location of target damage if any is reflected through TNKDAT.

'Shocked' is a temporary status wherein the tank is not able to respond to your command for one cycle of time. The 'shocked' status removal is directly done through the main programme.

3.3 How to use the Simulator:

There are 2 programmes in the system which are to be used in the following sequence.

(a) MAPRIT.PAS

(b) BATTLE.PAS

MAPRIT.PAS

The programme MAPRIT.PAS is used to describe the topography of the area that is being used. The following

data regarding the details to be described must be available at the time of execution.

(a) RIVERS/CANALS

i) Direction of flow of water

ii) Depth of water in cms

iii) Width in mtrs.

(b) OTHERS: For all others including the rivers and canals the following are required.

i) Start point

ii) Turning points

(c) CONTOURS: The height of the contour, in steps of 50 mtr, relative height with the surrounding area taken at '0'.

All the start points and turning points are to be given as 4 figure grid reference. In between these 'Turning Points' the particular detail is approximated as a straight line, hence the points have to be chosen accordingly. While describing the details restriction is also made for the direction in which the next grid point exist. The next grid point has to be in one of the 8 cardinal direction i.e. N/NE/E/SE/S/SW/W/NW.

When the next grid point has been wrongly directed, a message is given i.e. 'going out of map area - direction

wrong'. This point could be recalculated and given.

All contours must be complete i.e. open contours might lead to wrong result hence not allowed.

Any number of such details can be described but each one has to be described separately. For example a river and its tributary will be described separately.

Once this programme is executed two files 'MAPDAT' and 'MAP' are made. They can be used for any subsequent exercise till another map, is defined in a similar manner.

BATTLE.PAS: The execution of this programme will simulate the tank battle. The following data is to be kept ready at the start of the exercise and given when asked.

(a) Type of tank used: Each tank has particular data associated with it for its armour and guns. The following 2 tanks have been catered for:

i) Vijayanta

ii) T-55

(b) Location of Troops: Location of 5 troops of a squadron of tanks in 4 figure grid reference.

(c) Time: The time of start of simulation of the exercise is to be given. Along with it the time

step to be used for the exercise is to be given. The minimum time step is calculated as 1 min. This time step can be changed during the execution of the programme as well.

(d) Land marks: The land marks that are likely to be used during the exercise with their name (less than 12 chars without blanks) and its 4 figure grid reference. Any number of such land marks can be defined.

Once these details have been given the simulation of the exercise starts. There are various commands that can be given. There are basically 2 types of commands i.e. one that is addressed to a tank or otherwise.

The commands that could be addressed to a tank could be

a) To report about centain attributes

b) To move from one location to another

c) To fire at a location or tank.

These are given as Appx. 'A'. The other commands that could be given to the simulator are

(a) Help: To find the information regarding the commands addressable to the tanks.

(b) Check: The checking/changing of various status during the exercise like

i) Land marks

ii) Time step change

iii) List of tanks on the move

iv) Setting/Unsetting of map drawing.

(c) Freeze/Cont: This will freeze/unfreeze all the action part. This could be used at the time when some details have to be studied without change in status. 'Cont' will unfreeze the action.

(d) End: Used to terminate the programme.

Generally the following guidelines are to be used during interactive communication with the simulator.

(a) All sentences or where possibility of a sentence exists, the reply has to end with a '.'.

(b) A word is taken as 12 characters without blanks.

(c) The delimiters between words are 'blank/./,.

(d) All single word reply need not end with a '.'.

(e) For a Yes/No type of diaglogue, a simple 'Y' or 'N' could also be used.

The tank addressing is done by 'NO'. The 'NO' consists of 2 parts, Troop number followed by Tank number. Say there

are 5 troops and each troop having 3 tanks then TK 31 will mean Tank No. 1 of troop No. 3.

Normally map drawing will not be available. If this is required, then this will have to be set through CHECK. A one time map drawing is also available through this procedure.

CHAPTER IV

ALGORITHMS

In this chapter some algorithms will be discussed along with the assumptions/approximations that have been made.

4.1 Fire:

The procedure FIRE simulates the 'fire' part. The list of tanks that have been ordered to fire and details required is kept in a linked list of records FIRETAB. In each time step the number of shells that are to be fired is calculated, their effect analysed and reflected in the tank status of the affected tank. Once all the 'ordered' shells have been 'fired' by that tank then that record is deleted from the list.

The various fields in the record FIRETAB are as follows:

SOURCE TK:(INTEGER); Used for storing the number of the tank that is firing.

DEST TK :(INTEGER); Used to stores the number of the tank that is being fired upon. If it is '0' it indicates that firing is on a location.

SOURCE LOC: X,Y:(INTEGER): Stores the location co-ordinates of the tank firing the shells. Used for calculating distance and hit.

DEST LOC: X,Y:(INTEGER); Stores the location of tank being fired upon. Used for hit analysis and 'aiming'.

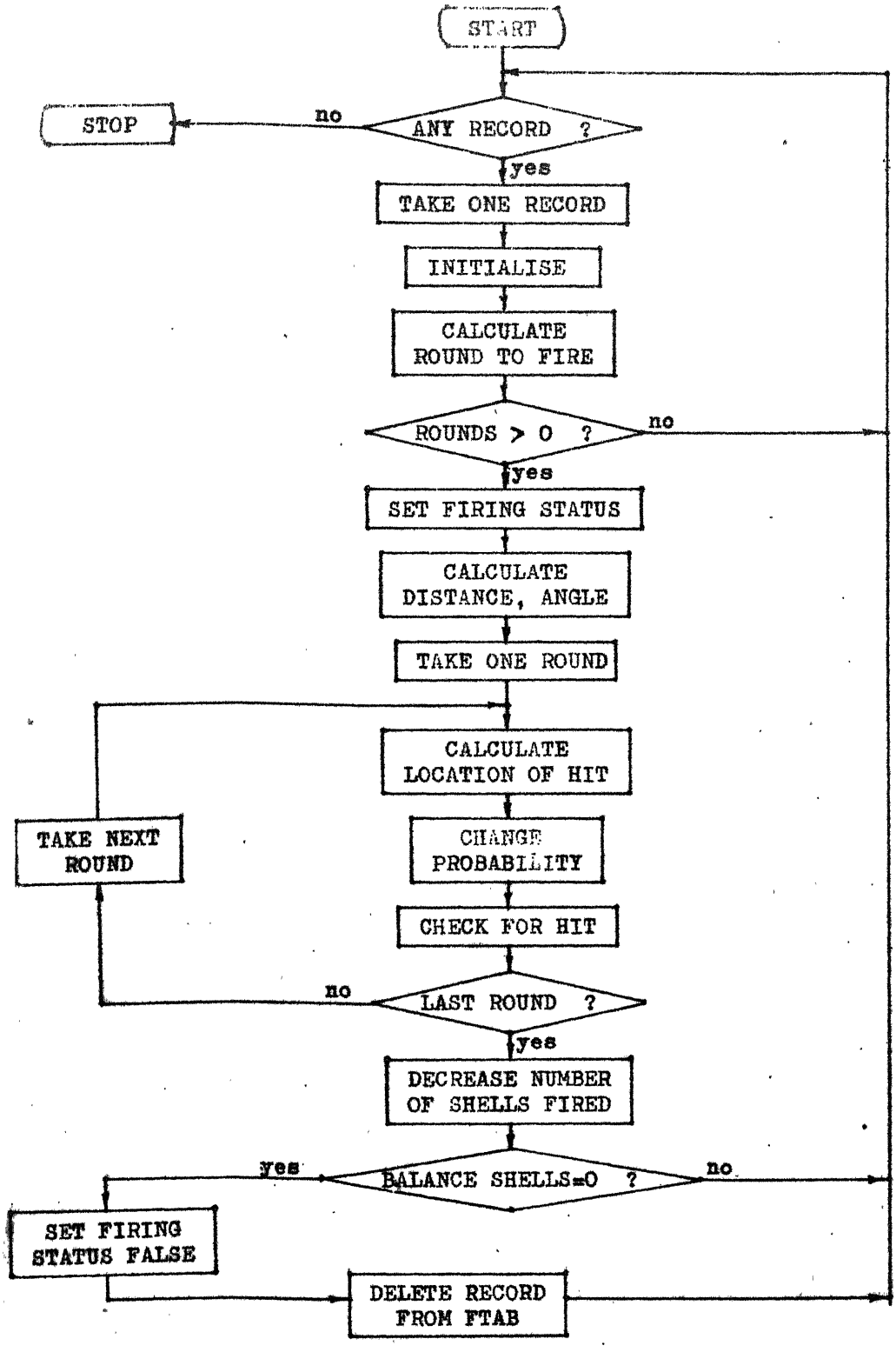
TOTNUM: (INTEGER); Gives the total number of shells that are left to be fired.

TIME: (INTEGER) ; This gives the load time i.e. time taken by the tank to initially start its first fire.

HIT PROB: (REAL) ; Gives the hit probability of the shell. Used for calculating the location of hit of the shell.

RATE : (0..15); This gives the number of shells that can be fired in a minute. This depends on the status of tank like moving etc.

ULINK, LLINK:(FPTR); Used to form a linked list of all the tanks that are firing and random deletion on completion of 'fire'.


START
STOP
no
ANY RECORD ?
yes
TAKE ONE RECORD
INITIALISE
CALCULATE ROUND TO FIRE
ROUNDS > 0 ?
no
yes
SET FIRING STATUS
CALCULATE DISTANCE, ANGLE
TAKE ONE ROUND
CALCULATE LOCATION OF HIT
CHANGE PROBABILITY
CHECK FOR HIT
TAKE NEXT ROUND
no
LAST ROUND ?
yes
DECREASE NUMBER OF SHELLS FIRED
yes
BALANCE SHELLS=0 ?
no
SET FIRING STATUS FALSE
DELETE RECORD FROM FTAB

F I R E

ALGORITHMS

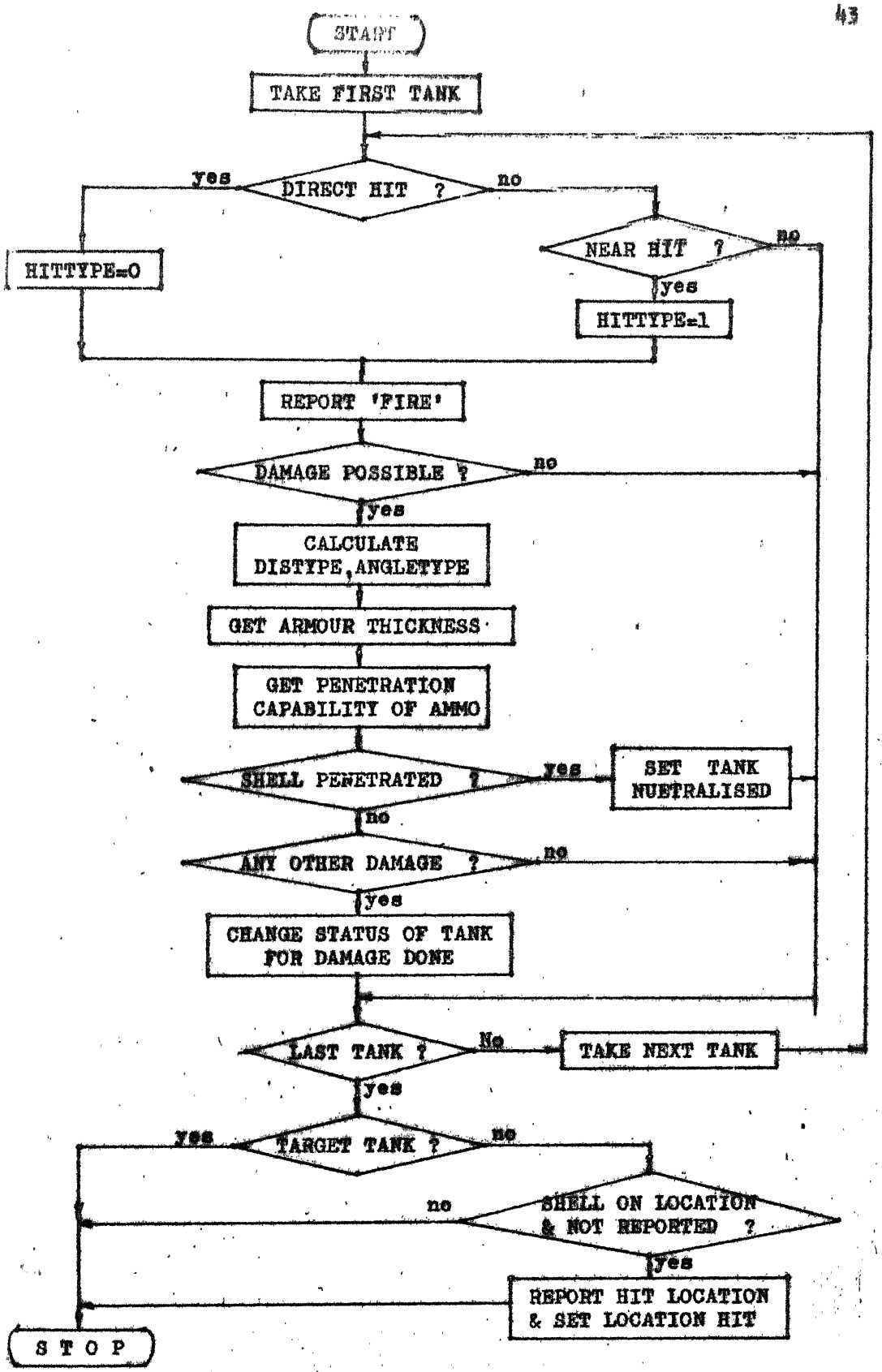
FIRE

```
1. FOR EACH RECORD IN LIST FTAB DO
   BEGIN
   2. INITIALISE
   3. CALCULATE NUMBER OF ROUNDS TO BE FIRED IN THIS CYCLE
   4. IF NUMBER OF ROUNDS GREATER THAN ZERO THEN
      BEGIN
      5. SET STATUS OF THAT TANK TO FIRING
      6. CALCULATE DISTANCE & ANGLE TO TARGET
      7. FOR EACH ROUND DO
         BEGIN
         8. CALCULATE LOCATION OF HIT
         9. CHANGE PROBABILITY OF HIT
         10. CHECK HIT AND ANALYSE ITS EFFECT
         END
      11. DECREASE NUMBER OF SHELLS FROM BALANCE
         12. IF BALANCE OF SHELLS TO BE FIRED EQUALS ZERO THEN
         BEGIN
         13. SET FIRING STATUS TO FALSE
         14. DELETE THIS RECORD FROM FIRETAB
         END
      END
END
```

CHECK HIT: Checks if any of the tanks have been hit directly or indirectly. If it is hit then calls for analysis for damage. Also checks, if the target is a location, then if it has been hit or not.

```
1. FOR EACH TANK DO
   BEGIN
   2. IF HIT LOCATION = TANK LOCATION THEN
      ANALYSE HIT FOR DIRECT HIT
   3. IF HIT LOCATION NEXT TO TANK LOCATION
      THEN ANALYSE HIT  FOR INDIRECT HIT
   END
4. IF TARGET IS NOT TANK THEN
   BEGIN
   5. IF HIT LOCATION = TARGET LOCATION AND
      HAS NOT BEEN REPORTED THEN
      BEGIN
      6. REPORT LOCATION HIT
      7. SET THIS TO REPORTED
      END
   END
```

ANALYSE HIT: Reports target tank coming under fire and then analyses the hit for any damage it may have done and changes the status if required.


START
TAKE FIRST TANK
DIRECT HIT ?
yes
no
HITTYPE=0
NEAR HIT ?
no
yes
HITTYPE=1
REPORT 'FIRE'
DAMAGE POSSIBLE ?
no
yes
CALCULATE DISTYPE,ANGLETYPE
GET ARMOUR THICKNESS
GET PENETRATION CAPABILITY OF AMMO
SHELL PENETRATED ?
yes
SET TANK NUETRALISED
no
ANY OTHER DAMAGE ?
no
yes
CHANGE STATUS OF TANK FOR DAMAGE DONE
LAST TANK ?
No
TAKE NEXT TANK
yes
TARGET TANK ?
yes
no
SHELL ON LOCATION & NOT REPORTED ?
no
yes
REPORT HIT LOCATION & SET LOCATION HIT
STOP

C H E C K H I T

1. IF NOT REPORTED THEN REPORT COMING UNDER FIRE
2. IF AMMUNITION IS NOT 'SMOKE' OR DISTANCE WITHIN RANGE

   BEGIN

   3. CALCULATE DIST-TYPE & ANGLE OF HIT ON THE TANK
   4. GET ARMOUR THICKNESS AT THAT POINT
   5. IF THICKNESS≤ PENETRATING CAPABILITY

      THEN SET TANK NUETRALISED

      ELSE
   6. ANALYSE FOR EXTENT OF DAMAGE TO TANK AND CHANGE

      STATUS ACCORDINGLY

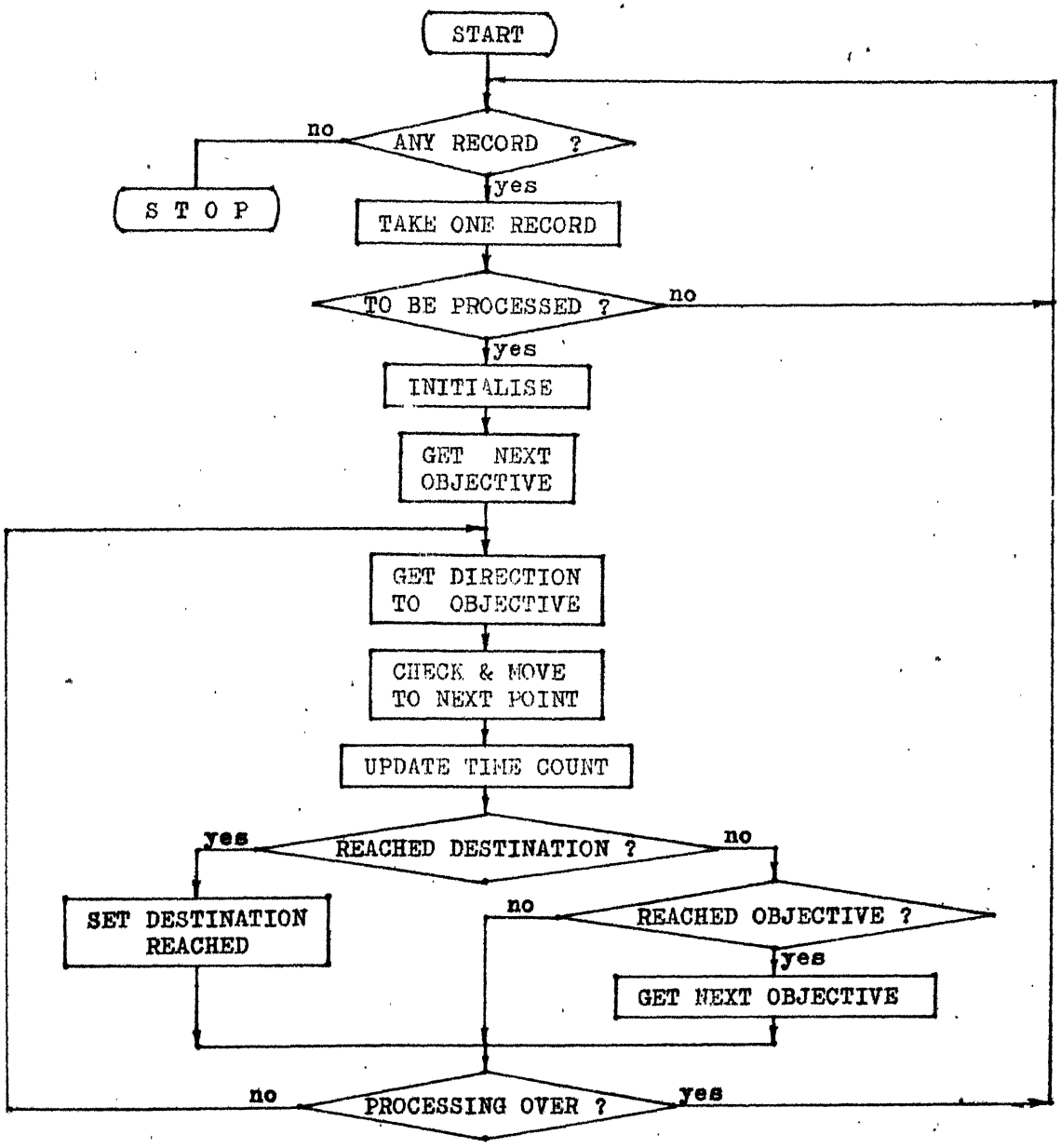
   END

4.2 MOVE TANKS:

This procedure simulates the 'move' part. The details regarding all the tanks that have been ordered to 'move' are kept in a linked list of records in MOVTAB.

In each cycle, each tank in this list is moved for a time period of one time step unless it comes to a halt in between. During the move it 'jumps' from one grid point to another. Before going to the next point it checks for its topography and its possibility to move to that location.

If some obstacle is encountered then it takes action depending on the type of obstacle and command given. The change of status if any is reflected. Once the destination


START
ANY RECORD ?
no
yes
S T O P
TAKE ONE RECORD
TO BE PROCESSED ?
no
yes
INITIALISE
GET NEXT OBJECTIVE
GET DIRECTION TO OBJECTIVE
CHECK & MOVE TO NEXT POINT
UPDATE TIME COUNT
yes
REACHED DESTINATION ?
no
SET DESTINATION REACHED
no
REACHED OBJECTIVE ?
yes
GET NEXT OBJECTIVE
no
PROCESSING OVER ?
yes

M O V E T A N K S

is reached or has been nuetralized or incapacitated permanently then this record is deleted from the move list MOVTAB.. The various fields in the record MOVTAB are

TANK NO: (0..99); The number of tank on the move.

SOURCE : X,Y:(INTEGER); The location of the starting point. X and Y cordinates of the tank.

DEST: X,Y: (INTEGER); The location it has to reach ultimately.

LOC: X,Y: (INTEGER); The present location where the tank is located.

ROUTE:ARRAY(1..10) OF CORD:X,Y:(INTEGER); Gives location of the intermediate points.

STATE:(BOOLEAN); Whether the tank is moving or stopped temporarily.

OBJECTIVE:(INTEGER): The total No. of intermediate points.

COURROBJECTIVE:(INTEGER); Present No. of the objective.

ALGORITHMS

MOVE

1. FOR EACH RECORD IN LIST DO

   BEGIN

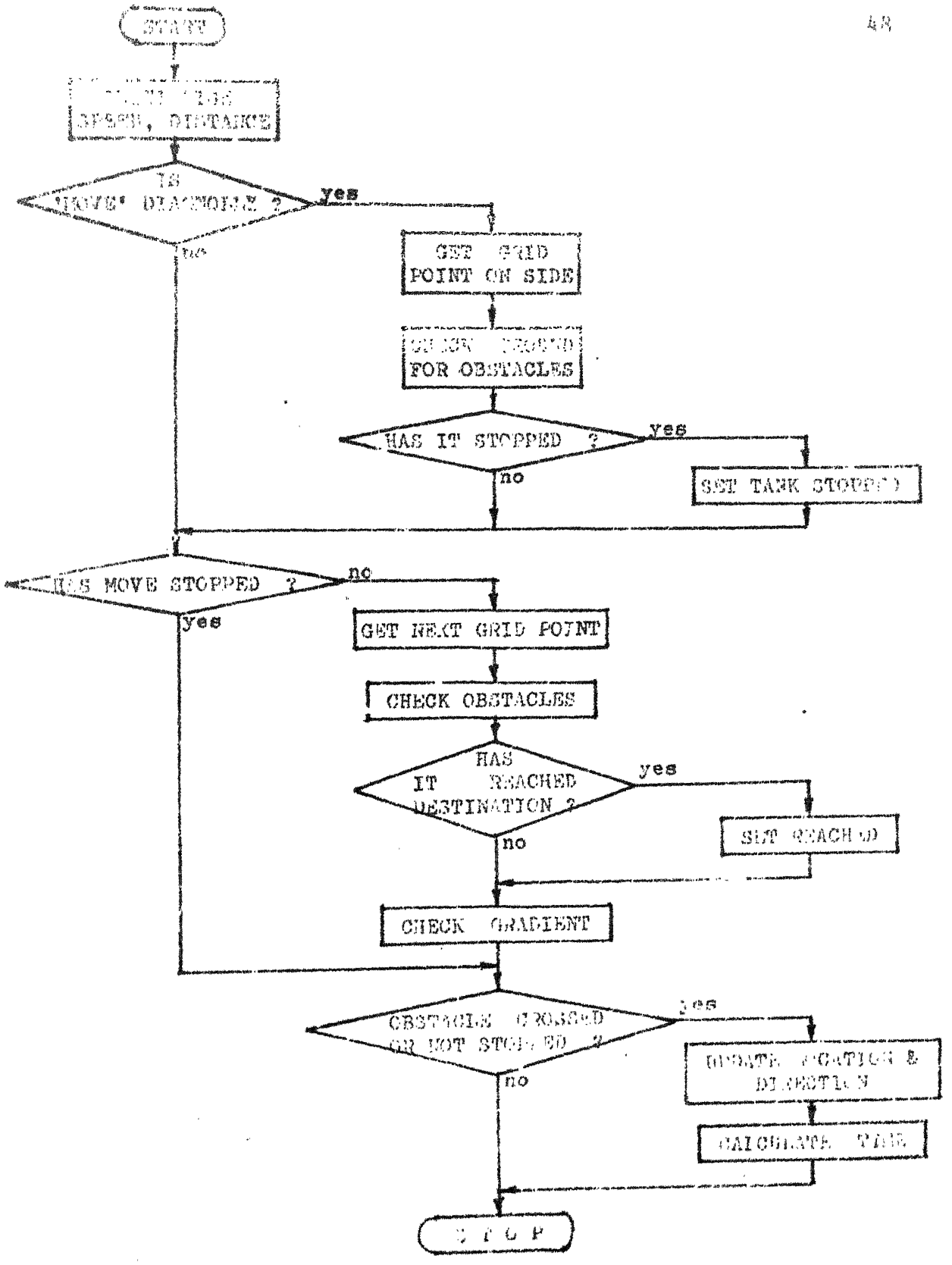
```
2. IF RECORD REQUIRES PROCESSING THEN
   BEGIN
   3. INITIALISE
   4. GET NEXT OBJECTIVE
      REPEAT
      5. GET DIRECTION TO OBJECTIVE
      6. COMPUTE TIME FOR MOVE TO NEXT POINT
         IF MOVE IS POSSIBLE
      7. INCREMENT TIME COUNT WITH CURRENT TIME
      8. IF REACHED DESTINATION,SET DESTINATION  REACHED
      9. ELSE REACHED OBJECTIVE THEN GET
         NEW OBJECTIVE
      UNTIL REACHED DESTINATION OR FINISHED
            MOVE OR TIME STEP OVER
   END
END
```

COMPUTE TIME

This procedure calculates the next point that the tank is to be moved. Then checks for the type of ground that is there between the 2 points and the next point. If it can move to the next point then the following is calculated.

a) Speed in this terrain

b) Distance to the point



COMPUTE TIME

c) Gradient between the points

d) Fuel consumed

If it is not possible to move directly then checks if it can encounter this obstacle taking into consideration the following

a) Type of obstacle

b) Capability of the tank

c) If water obstacle, then reports this and asks for orders. If ordered to cross then checks if it can, otherwise gets bogged.

COMPUTE TIME

```
1. INITIALISE DATA
2. IF TANK TRAVELLING DIAGONALLY THEN CHECK
   GROUND IN BETWEEN FOR OBSTACLES
3. IF MOVE NOT FINISHED THEN
   BEGIN
   4. GET NEXT GRID POINT
   5. CHECK GROUND FOR OBSTACLES
   6. IF GRADIENT WITHIN LIMIT THEN
         BEGIN
         7. CALCULATE GRADIENT
         8. CALCULATE SPEED CHANGE
```

```
        9. FUEL CONSUMPTION CALCULATE
        END
    ELSE
    10. STOP TANK AT LOCATION
END
11. IF TANK HAS NOT STOPPED OR HAS CROSSED OBSTACLE
BEGIN
    12. MOVE TANK TO NEXT LOCATION
    13. UPDATE DIRECTION OF TANK
    14. CALCULATE TIME TAKEN TO REACH NEXT POINT
    15. CALCULATE FUEL CONSUMED AND
        DECREASE FROM BALANCE
END
```

4.3 Approximation/Assumptions:

For the simulation process some approximations and assumptions are made and these have been done keeping in view the realistic situation and its application on the computer. Some of these have been described in the succeeding paragraphs.

Distance between grid points is kept at 500 meters due to the following reasons:

(a) Depiction of a large area of 20x20 Kms used for the mobility of the tank.

(b) Decreasing the number of grid point for the same area thereby decreasing the memory requirements.

(c) Resolution in graphic display due to limitation of size of display which is fixed.

Straight line approximation between points. In this case error is maximum within 500 mtr length because any number of turning points can be described. The limitation in direction restricted to only 8 is a constraint mainly for storage of data and computation simplicity. The deviation to some extent is acceptable for the exercise purpose.

Only six details like river, canal, road etc. have been included for the map description. Other details which do not have much bearing on the tank battle like anti-personnel mines, barbed wire fence, booby traps etc. have not been considered.

The minimum time step has been kept as 1 minute taking into consideration the maximum time taken by a tank to move from one point to another in the worst possible condition.

The Move and Fire operations are done serially one followed by the other and not parallel as it happens in real time. This is due to computer programme control flow being sequential.

The 'shocked' tank is kept under shock only for one cycle of time but in actual practice this could be of variable length of time but is approximated as one cycle.

During the firing of shells, the rest of action is frozen as the time of flight is considered very small compared to movement during that period of time.

The height between 2 contours is kept constant throughout this area and height changes only from the next contour onwards.

4.4 Improvements:

In the 'MOVE' commands more general commands like move in formation, and deploy various tactical formation can be adopted and catered for.

Different type of firing tactics can be catered for. Here only 'Ranging' is used by SQUARE ROOT method.

More accurate analysis of hit of the shell could be done. In this, approximation is done upto an accuracy of ±250 meters which may not be accurate in certain cases.

More details in the map description could be given like tracks, rough or forested area or undulating ground as found in deserts which provide cover to tanks or obstruct visibility could be described and used.

To give more realistic view, Air threat, Artillery and Anti-tank weapons could be incorporated in the simulations.

A combination of MOVE & FIRE tactics used in the case of providing covering fire for an attack could be included.

4.5 Time & Memory Considerations:

The simulator programme is written in Pascal language and implemented on system Dec 1090 computer. All the timings and memory requirement are corresponding to this system.

(a) Programme Storage

BATTLE.PAS	116K bytes
MAPRIT.PAS	20K bytes

(b) Compilation data using

GPAS (Graphic Pascal)

Memory requirement = 105K bytes

Run time = 11 secs

(c) Programme Execution

i) Time for initialisation, including reading of map, tank locations, land marks Time etc. = 10 secs (average).

ii) Command Processing Times

Simulator command = 33 ms (average)

Tank commands = 16 ms (average)

iii) Action Processing time = 26 ms (average)

iv) Memory required during execution = 275K bytes (for programme example)

It must be noted that these timings and memory requirements will vary depending on the simulated exercise as dynamic storage and processing is used by the programme.

(d) Type of computer required: For the running of this programme any general purpose computer having the following facilities could be used.

i) Memory of 64K

ii) Terminals (TTY) for interactive usage

iii) Graphic Terminal device (Tectronix display terminal) for the use of displaying of map if required.

iv) Availability of Pascal graphics with its linking with GPGS.

CHAPTER V

GRAPHIC AIDS

The graphic aid used here for display purpose is Graphic Terminal (Tectronix display terminal) and the language used is Pascal (Graphics).

The size of display on this device is limited to physical size of 15 cms x 15 cms. On this is displayed an area of 20x20 kms divided into 40x40 grid points.

For easy referencing on the map '+' markers are displayed at an interval of 5 grid points i.e. distance of 2.5 kms. each.

The legend for the details represented with the markers that have been used are

a) RIVER (double line) =

b) CANAL (nabla) f

c) ROAD (straight line) -

d) MINEFIELD (cross) X

e) MARSHY GROUND (Diamond) ◇

f) GRASS/FIELD (Delta) △

g) CONTOURS (dot) .

The location of hit of a shell is displayed by a * at the location of hit. This is done only for those shells that

are fired in that cycle only. This is done by storing the hit location in a file FIREDT when the Fire procedure is executed. This data is used later by the programe DRAW MAP while drawing the map.

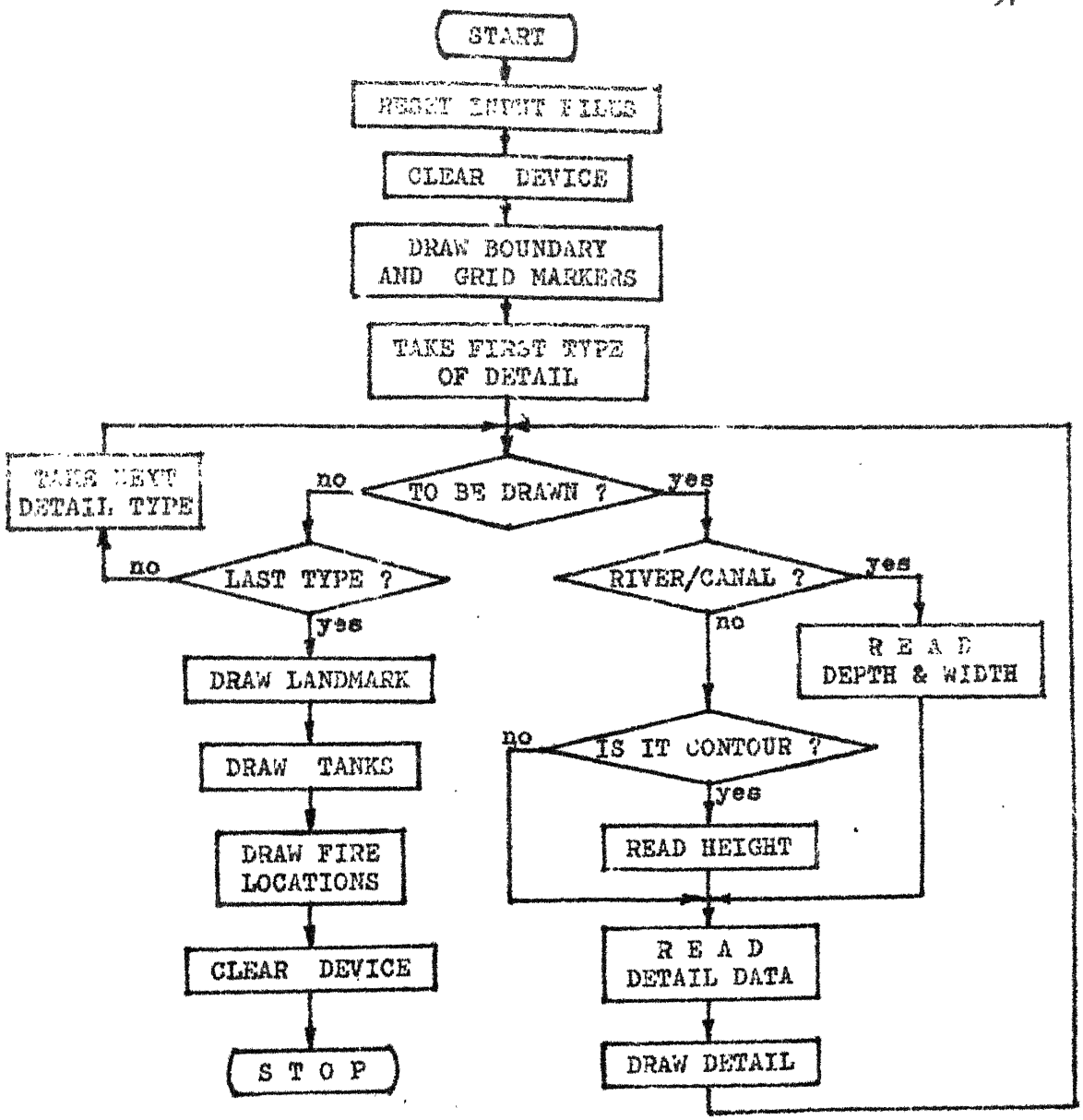
All land marks as described by the user is displayed on the terminal by the name at its location. Its exact location is given by starting character of the land mark name.

All tanks are displayed merely by their numbers i.e. Tank No. 3 of troop No. 1 is displayed as 13 on the displayed. The starting number gives the exact location of the tank.

The programme for Graphic Display is written in Pascal which does not at present provide all the facilities that are otherwise available in Fortran (GPGS). It is also handicapped by the lack of literature on it. Because of these restrictions graphic display is also restricted. This could be developed further as and when the Graphic Pascal becomes more powerful.

5.1 DRAW MAP:

This is the procedure used for drawing the map on the graphic terminal. It takes the input data from files MAPDAT, MAP and FIREDT. Apart from this the data for tanks location


START
RESET INPUT FILES
CLEAR DEVICE
DRAW BOUNDARY AND GRID MARKERS
TAKE FIRST TYPE OF DETAIL
TAKE NEXT DETAIL TYPE
no
TO BE DRAWN ?
yes
no
LAST TYPE ?
yes
RIVER/CANAL ?
yes
no
R E A D DEPTH & WIDTH
DRAW LANDMARK
DRAW TANKS
DRAW FIRE LOCATIONS
CLEAR DEVICE
S T O P
no
IS IT CONTOUR ?
yes
READ HEIGHT
R E A D DETAIL DATA
DRAW DETAIL

DRAW MAP

is taken from Record TNKDAT and for land marks from LMTAB. The flow chart for this is shown as Figure 6.

ALGORITHM

1. RESET INPUT FILES MAPDAT, MAP AND FIREDT
2. CLEAR GRAPHIC DEVICE
3. DRAW BOUNDARY AND GRID MARKERS
4. FOR EACH TYPE OF DETAIL DO
   BEGIN
   5. WHILE DETAILS OF THIS TYPE ARE TO BE DRAWN DO
      BEGIN
      6. IF DETAIL IS RIVER OR CANAL READ DEPTH & WIDTH
      7. ELSE IF DETAILS IS CONTOUR READ HEIGHT
      8. READ OTHER DATA REGARDING DETAIL
      9. DRAW DETAIL ON DEVICE
      END
   END
10. DRAW LANDMARKS
11. DRAW TANKS
12. DRAW FIRED SHELL LOCATIONS
13. CLEAR DEVICE IF SEEN

5.2 Test Exercise on Simulator:

For the purpose of demonstration, as to how this simulator could be used, a small test exercise is demonstrated

as follows. Assymptions made for this purpose and a narrative are given below. The aids used for this purpose are, the Hard Copy terminal for recording the interactive communication dialogues between the user and simulator and the 'plotter' to get the map output.

Exercise Background:

There are two states OWN DESH and ENEMY DESH, which have unfriendly relations. The NULLAH river which flows from loc 0127 to loc 3907 is the international boundary between them. Own Desh consists of troops 1,2 and 3 and Enemy Desh has troops 4 and 5. Their locations and various LANDMARKS are shown in MAP-1. Area HUT being a strategically important location Own Desh decides that it must capture it. It ventures to do this in the following manner.

(a) AIM: Troop 1 and Troop 2 to capture area HUT by 042300 Hrs.

(b) PLAN: The plan to capture HUT is to be done in 2 phases.

i) Phase 1: To cross river NULLAH and make a bridge, Troop 1 in area TREE and Troop 2 in area BRIDGE.

ii) Phase 2: Troop 2 to move and capture area NODE to give support to Troop 1 for the capture of area HUT.

(c) PROGRESS OF BATTLE: The course of events that take place during this operation are as follows:

i) Because of the increasing tension, Enemy Desh sends TK 53 to 0723 and TK 42 to 3117 for observation.

ii) To divert attention Troop 1 and 2 fire smoke shells at loc 0823 and 3113 to give the impression that a river crossing and attack is being planned in this area.

iii) Falling for this TK 51 and TK 41 are sent by Enemy Desh to locs 0822 and 3113.

iv) Troops 1 and 2 cross the river in area CROSS and BRIDGE respectively.

v) Troop 3 is brought up to Loc 1717 to give support to these troops.

iv) TK 52 moves to 1727 and TK 51 and TK 53 try and cross the NULLAH river to move from the back.

vii) Troop 2 moves towards NODE and after softening it with fire assaults on it to capture it.

viii) Troop 1 after nuetralising TK 52 moves to area HUT.

(d) The activity recorded on the map's correspond to starting of the exercise (MAP-1), after completion of phase 1 (MAP-2) and after completion of the exercise (MAP-3).

(e) The interactive dialogue used for this exercise is given as Appendix 'B'.

CHAPTER VI

CONCLUSION

A simulator for tactical exercise based on tank has been described here in reasonable detail. This simulator takes into account only a chosen set of factors and follows a specific set of rules built into it. Various constants such as those describing the accuracy of armament, the probability of nuetralization and various kinds of hits are used by the simulator.

It is hoped that this simulator will form a logical extension of a sand model exercise, so that it provides more realistic details such that it gets close to real exercises in field. It will enable exercises in critical areas, such as enemy held territory or civilian territory in accessable for real exercises. By co-ordinating intelligence information with known topographical details of critical territories very effective familiarization with such areas can be ensured by simulator exercises.

The simulator goes beyond Armour battle and is very effective in training human decision making element in controlling complex operations, the outcome of which is primarily decided by equipment characteristics, fire power and tactics employed.

Man-machine communication is an important aspect which has been dealt with in this simulation. It is hoped that this will provide to the user how best to use this effectively to deal with a variety of command and control tasks.

It is considered that this simulator could be a useful training aid for Armoured Corp Officers in learning squadron level tactics. The simulator could be used both as a programmed teaching aid for minor tactics as well as training the squadron as a team.

6.1 Further Development:

A small operating system to simulate a squadron level battle to include the following features could be added.

a) 2 rival squadrons are able to operate independently.

b) Classification of messages, so that they are routed to various combination of leaders/tanks.

c) An umpire who gets all messages and commands given by all originators.

d) Partial display of the areas under their squadron to the Squadron commander and troop commanders.

e) An element of chance to intercept messages of the opposing forces.

REFERENCES

1. Capt. Gurmit Singh, 'War Gaming', M.Tech. Aug. 1979.

2. Maj. S.C. Gupta and S. Ramani, 'Computers and Tactical War Gaming', Technical Report No. 44, TIFR, June 1968.

3. Richard F. Barton, 'A Primer on Simulation and Gaming', Prentice-Hall Inc.

4. Melvin Dresher, 'Game of Strategy-Theory and Applications', Prentice-Hall, Inc.

5. George W. Evans, Graham F. Wallace, Georgia L. Sutherland, 'Simulation using Digital Computers,' Prentice-Hall Inc.

APPENDIX A

COMMANDS AVAILABLE WITH OPTIONS

COMMAND:: = TK (TANK NUMBER) (REPORT PART)/(FIRE PART)/(MOVE PART).

REPORT PART:: = REPORT LOCATION

STATUS TK (TANK NUMBER)/$ LOC (GRID REF)/$

ENEMY

BALANCE AMMUNATION/FUEL HE/SMOKE/AP/ALL/$

VISIBILITY TK(TANK NUMBER)/LOC (GRID REF)

FIRE PART:: = STOP FIRING

FIRE/SHOOT (NO) HE/AP/SMOKE SHELL(S) AT TK (TANK NUMBER)(DIRECTION)
LN (GRID REF)

MOVE PART:: = MOVE TO (LAND MARK) /$

ROUTE (LAND MARKS)

(DISTANCE) MTR/METER/KM (DIRECTION)

OF (LAND MARK)

ROUTE (LAND MARK)

NOTE:

TANK NUMBER :: = TROOP NO. TANK NO.
DIRECTION :: = CLOCK DIRECTION/CARDINAL DIRECTION/DEGREE
CARDINAL DIRECTION:: = NORTH/N, NORTH EAST/NE etc.
LAND MARK :: = DEFINED LAND MARK/GRID REF
LAND MARKS :: = LAND MARK, LAND MARKS/$
GRID REF :: = FOUR FIGURE GRID REFERENCE

GPGS-10 VERSION-001

THIS IS A PROGRAMME EXERCISE FOR TANK BATTLE AT SQUADRON LEVEL. YOU
ARE REQUIRED TO TYPE IN COMMAND IN THE FORMAT SPECIFIED AND IT WILL BE
EXECUTED. THE FOLLOWING POINTS NEED PARTICULAR ATTENTION:-
1 THE COMMAND MUST BE TYPED STARTING WITH <TK> OR OTHER COMMAND
2 THE COMMAND MUST END WITH <.>
3 IN CASE OF HELP TYPE <HELPCOMMD> OR <HELP>
4 TO TERMINATE THE PROGRAMME TYPE <END.>
5 AFTER YOU HAVE WRITTEN YOUR COMMAND GIVE CARRAIGE RETURN
6 IF YOU WANT THESE COMMANDS TO TYPED THEN TYPE COMMAND <CHKPTS.
READY TO START NOW
PLEASE GIVE THE TYPE OF TANK USED -
if it is VIJAYANTA then type "V"
if it is T-55 then type "T"
type of tank :V
PLEASE GIVE THE LOCATION OF THE VARIOUS TROOPS OF YOUR SQUADRON
YOU ARE REQUIRED TO GIVE ONLY THE FOUR FIGURE GRID REFERENCE
TROOP NO 1:1519
TROOP NO 2:2511
TROOP NO 3:1511
TROOP NO 4:3133
TROOP NO 5:0531
IMPORTANT:note that the map details have been taken from your file ="MAP

DO YOU WANT TO DEFINE LANDMARKS ?Y
TYPE IN LANDMARKS IN THE FOLLOWING FORMAT [<LM> <GR>.]
WHEN YOU HAVE FINISHED then type [FINISH.]
INPUT:HILL 1509.
INPUT:RDJN 2505.
INPUT:CROSS 1818.
INPUT:BRIDGE 2513.
INPUT:NULLAH 1021.
INPUT:TREE 1523.
INPUT:HUT 2129.
INPUT:NODE 2929.
INPUT:FIELD 1733.
INPUT:FINISH.
DO YOU WANT THE TABLE OF LANDMARKS ?N
DO YOU WANT TO CHANGE TIMESTEP? [BY DEFAULT IT IS TAKEN AS 1 MIN] :Y

```
PLEASE TYPE THE TIMESTEP IN FORMAT [<TIME> HR/MIN/SEC ,] :3 MIN.
GIVE TIME OF START OF EXERCISE IN (HR,MIN) 4 FIGURES:1900

YOUR COMMAND:TK 42 MOVE TO 3117 ROUTE 3529,3519.
                                                    TIME:19: 3: 0
ROGER!



YOUR COMMAND:TK 53 MOVE TO 0723 ROUTE 0425.
                                                    TIME:19: 6: 0
ROGER!



TANK  4 2  REPORTING FROM LOCATION  35 24  CANAL  OBSTACLE AHEAD
with MAIN CHARACTEREstics AS FOLLOWS:=

                          DIRECTION   SOUTH WEST  =>>   NORTH EAST
APPROX WIDTH     50  METERS
APPROX DEPTH     25  CMS
SHOULD I NEGOTIATE THE OBSTACLE or CHANGE COURSE ?
TYPE IN a "C" FOR CHANGING COURSE OR A "GO" TO CONTINUE:GO
TANK  4 2 REPORTING > CANNOT NEGOTIATE THE CANAL - SIDE ARE VERY STEEP


TK  5 3 REPORTING REACHED DESTINATION  723
YOUR COMMAND:TK 13 FIRE 6 SMOKE SHELL LN 0823.
                                                    TIME:19: 9: 0

TK 13 REPORTS GIVING EFFECTIVE FIRE ON LN  823
YOUR COMMAND:TK 22 FIRE 6 SMOKE SHELL LN 3113.
                                                    TIME:19:12: 0

TANK 22 REPORTS GIVING EFFECTIVE FIRE ON LN 3113
YOUR COMMAND:TK 41 MOVE TO 2713 ROUTE 3527,3525,2725.
                                                    TIME:19:15: 0

ROGER!



YOUR COMMAND:TK 51 MOVE TO 0822 ROUTE 0425.
                                                    TIME:19:18: 0

ROGER!
```

```
TK 5 1 REPORTING REACHED DESTINATION  822
YOUR COMMAND:FREEZE.
                                                    TIME:19:21: 0
YOUR COMMAND:TK 13 MOVE TO 1922 ROUTE CROSS.
                                                    TIME:19:21: 0
ROGER!

YOUR COMMAND:TK 12 MOVE TO 1624 ROUTE  TREE.
                                                    TIME:19:21: 0
ROGER!

YOUR COMMAND:TK 23 MOVE TO 2517.
                                                    TIME:19:21: 0
ROGER!

YOUR COMMAND:TK 22 MOVE TO 2717 ROUTE BRIDGE.
                                                    TIME:19:21: 0
ROGER!

YOUR COMMAND:TK 32 MOVE TO 1717 ROUTE 1910,2111.
                                                    TIME:19:21: 0
ROGER!

YOUR COMMAND:TK 33 MOVE TO 1517 ROUTE 0911.
                                                    TIME:19:21: 0
ROGER!

YOUR COMMAND:TK 11 FIRE 8 HE SHELL LN 1123.
                                                    TIME:19:21: 0
YOUR COMMAND:TK 21 FIRE 8 HE SHELL LN 2717.
                                                    TIME:19:21: 0
YOUR COMMAND:CHECK.
                                                    TIME:19:21: 0
THE FOLLOWING CHECKS ARE POSSIBLE
                  LISTING LANDMARKS TYPE "L"
                  LISTING CURRENTLY MOVING TANKS-"T"
                  DRAWINGG THE MAP "M"
                  CHANGING TIMESTEP & TIME "C"
                  SETTING TO DRAW THE MAP-"S"
                  DELETING THE SET MAP DRAWING-"D"
                  RETURN-"R"
CHECK REQUIRED FOR:C
DO YOU WANT TO CHANGE TIME STEP? [BY DEFAULT IT IS TAKEN AS 1 MIN] :N
```

GIVE TIME OF START OF EXERCISE IN (HR,MIN) 4 FIGURES:1921
YOUR COMMAND:CONT.

TIME:19:21: 0

TANK 1 2 REPORTING FROM LOCATION 16 19 RIVER OBSTACLE AHEAD
WITH MAIN CHARACTERISTICS AS FOLLOWS:-

DIRECTION NORTH WEST =>> SOUTH EAST
APPROX WIDTH 100 METERS
APPROX DEPTH 75 CMS
SHOULD I NEGOTIATE THE OBSTACLE OR CHANGE COURSE ?
TYPE IN A "C" FOR CHANGING COURSE OR A "GO" TO CONTINUE:GO

TANK 2 3 REPORTING FROM LOCATION 24 13 RIVER OBSTACLE AHEAD
WITH CHARACTERISTICS AS FOLLOWS:-

DIRECTION NORTH WEST =>> SOUTH EAST
APPROX WIDTH 100 METERS
APPROX DEPTH 75 CMS
SHOULD I NEGOTIATE THE OBSTACLE OR CHANGE COURSE ?
TYPE IN A "C" FOR CHANGING COURSE OR A "GO" TO CONTINUE:GO

YOUR COMMAND:TK 52 MOVE TO 1728 ROUTE 0534,1734.

TIME:19:43: 0

ROGER!

TANK 11 REPORTS GIVING EFFECTIVE FIRE ON LN 1123
TANK 21 REPORTS GIVING EFFECTIVE FIRE ON LN 2717
YOUR COMMAND:TK 53 MOVE TO 0919 ROUTE 0420.

TIME:19:44: 0

ROGER!

TK 1 2 REPORTING REACHED DESTINATION 1624

TK 2 3 REPORTING REACHED DESTINATION 2517

TK 32 REPORTING REACHED DESTINATION 1717

TANK 5 3 REPORTING FROM LOCATION 7 23 RIVER OBSTACLE AHEAD
with MAIN CHARACTEREstics AS FOLLOWS:=

```
                          DIRECTION   NORTH WEST  -->   SOUTH EAST
APPROX WIDTH    100  METERS
APPROX DEPTH     75  CMS
SHOULD I NEGOTIATE THE OBSTACLE or CHANGE COURSE ?
TYPE IN A "C" FOR CHANGING COURSE OR A "GO" TO CONTINUE:GO
TANK 11 REPORTS GIVING EFFECTIVE FIRE ON LN 1123
TANK 21 REPORTS GIVING EFFECTIVE FIRE ON LN 2717
YOUR COMMAND:TK 11 FIRE 8 AP SHELL AT TK 51.
                                                  TIME:19:45: 0




TK  3 3 REPORTING REACHED DESTINATION 1517


YOUR COMMAND:TK 03 FIRE 3 HE SHELL LN 2725.
                                                  TIME:19:46: 0


TK   1 3 REPORTING REACHED DESTINATION 1922

TK   2 2 REPORTING REACHED DESTINATION 2717


YOUR COMMAND:CONT.
                                                  TIME:19:47: 0
```

```
TANK 5 3 REPORTING >  FROM LOCATION  620  TANK BOGGED DOWN IN MARSHY GR
TANK 23 REPORTS GIVING EFFECTIVE FIRE ON LN 2725
YOUR COMMAND:TK 23 REPORT VISIBILITY TK 41.
                                                    TIME:19:48: 0

TANKLOCATION        27          20
CANNOT SEE THAT DISTANCE AT NIGHT
IS NOT VISIBLE TO ME
YOUR COMMAND:TK 23 FIRE 4 AP SHELL LN 2722.
                                                    TIME:19:49: 0


YOUR COMMAND:TK 22 FIRE 4 HE SHELL AT TK 41.
                                                    TIME:19:50: 0




YOUR COMMAND:CONT.
                                                    TIME:19:51: 0


TK  4 1 REPORTING REACHED DESTINATION 2713



TK  5 2 REPORTING REACHED DESTINATION 1728
YOUR COMMAND:TK 41 FIRE 5 HE SHELL LN 2617.
                                                    TIME:19:52: 0

YOUR COMMAND:CONT.
                                                    TIME:19:53: 0

TANK 41 REPORTS GIVING EFFECTIVE FIRE ON LN 2617
YOUR COMMAND:TK 23 FIRE 6 HE SHELL AT TK 41.
                                                    TIME:19:54: 0

YOUR COMMAND:CONT.
                                                    TIME:19:55: 0
```

```
TANK 41 REPORTING!UNDER ENEMY HE   FIRE FROM LOC 2517
TANK 23 REPORTS NUETRALISING ENEMY TK-> LOC 2713
YOUR COMMAND:FREEZE.
                                                    TIME:19:54: 0
YOUR COMMAND:TK 13 MOVE TO 1925.
                                                    TIME:19:56: 0
ROGER!

YOUR COMMAND:TK 12 MOVE TO 1927.
                                                    TIME:19:56: 0
ROGER!

YOUR COMMAND:TK 23 MOVE TO 2727.
                                                    TIME:19:56: 0
ROGER!

YOUR COMMAND:TK 22 MOVE TO 3133 ROUTE 2532.
                                                    TIME:19:56: 0
ROGER!

YOUR COMMAND:TK 32 FIRE 10 HE SHELL LN 0923.
                                                    TIME:19:56: 0
YOUR COMMAND:TK 11 MOVE TO 1923.
                                                    TIME:19:56: 0
ROGER!

YOUR COMMAND:CONT.
                                                    TIME:19:56: 0
```

```
TANK  1 2  REPORTING FROM LOCATION  16 24  RIVER  OBSTACLE AHEAD
with MAIN CHARACTEREstics AS FOLLOWS:=

                          DIRECTION   NORTH        =>>    SOUTH
APPROX WIDTH     25   METERS
APPROX DEPTH     25   CMS
SHOULD I NEGOTIATE THE OBSTACLE or CHANGE COURSE ?
TYPE IN A "C" FOR CHANGING COURSR OR A "GO" TO CONTINUE:GO
```

```
TANK  1 1  REPORTING FROM LOCATION 15 19 RIVER  OBSTACLE AHEAD
with MAIN CHARACTEREstics AS FOLLOWS:-

                          DIRECTION   NORTH WEST  =>>    SOUTH EAST
APPROX WIDTH    100    METERS
APPROX DEPTH     75    CMS
SHOULD I NEGOTIATE THE OBSTACLE or CHANGE COURSE ?
TYPE IN "C" FOR CHANGING COURSE OR A "GO" TO CONTINUE:C
TANK ALREADY ON MOVE. DO YOU WANT TO CHANGE STATUS ?Y
FOR CHANGING ONLY DESTINATION TYPE "D"
FOR CHANGING ONLY ROUTE TYPE "R"
FOR COMPLETE CHANGING "C"
FOR RETURN WITHOUT CHANGING "N"
TYPE OF CHANGE REQUIRED ?D
NEW DESTINATION:1925
NOW COMMAND:(CONT.
                                                    TIME:19:57: 0


TK  1 2 REPORTING REACHED DESTINATION 1927


TANK  1 1 REPORTING FROM LOCATION  15 20  RIVER  OBSTACLE AHEAD
with MAIN CHARACTEREstics AS FOLLOWS:-

                         DIRECTION   WEST          =>>    SOUTH EAST
APPROX WIDTH    100    METERS
```

APPROX DEPTH   75   CMS
SHOULD I NEGOTIATE THE OBSTACLE or CHANGE COURSE ?
TYPE IN A "C" FOR CHANGING COURSE OR A "GO" TO CONTINUE:GO
TANK 32 REPORTS GIVING EFFECTIVE FIRE ON LN   923
YOUR COMMAND:TK 13 FIRE 4 AP SHELL AT TK 52.

TIME:19:58: 0

TK 1 3 REPORTING REACHED DESTINATION 1925

TANK 32 REPORTS GIVING EFFECTIVE FIRE ON LN 0923
YOUR COMMAND:TK 52 FIRE 4 HE SHELL LN 1725.

TIME:19:59: 0

TK 1 1 REPORTING REACHED DESTINATION 1525

TANK 52 REPORTING!UNDER ENEMY AP   FIRE FROM LOC 1924
YOUR COMMAND:TK 21 MOVE 2 KM NORTH.

TIME:20: 0: 0

ROGER!

TANK 52 REPORTS GIVING EFFECTIVE FIRE ON LN 1725
YOUR COMMAND:TK 23 FIRE 8 HE SHELL LN 3133.

TIME:20: 1: 0

```
TK   2 3 REPORTING REACHED DESTINATION 2727


YOUR COMMAND:TK 12 FIRE 4 AP SHELL LN 1728.
                                                    TIME:20: 2: 0


TANK 43 REPORTING!UNDER ENEMY HE     FIRE FROM LOC 2625
TANK 23 REPORTS NUETRALISING ENEMY TK-> LOC 3032
TANK 23 REPORTS GIVING EFFECTIVE FIRE ON LN 3133
YOUR COMMAND:CONT.
                                                    TIME:20: 3: 0


TK   2 1 REPORTING REACHED DESTINATION 2515


TANK 43 REPORTING!UNDER ENEMY HE     FIRE FROM LOC 2625
TANK 23 REPORTS GIVING EFFECTIVE FIRE ON LN 3133


TANK 52 REPORTING!UNDER ENEMY AP     FIRE FROM LOC 1927
TANK 12 REPORTS GIVING EFFECTIVE FIRE ON LN 1728
YOUR COMMAND:TK 22 REPORT LOCATION.
                                                    TIME:20: 4: 0
TANK   2 OF TROOP  2    REPORTING

  LOCATION            X         Y
                      25        31


YOUR COMMAND:TK 23 MOVE TO 3133.
                                                    TIME:20: 5: 0
```

ROGER!

TANK 2 3 REPORTING > FROM LOCATION 2828
  TANK HAS RUN INTO ENEMY MINEFEILD!!
YOUR COMMAND:TK 13 FIRE 4 SMOKE SHELL LN 2129.

TIME:20. 6: 0

YOUR COMMAND:TK 12 MOVE TO 2130.

TIME:20: 7: 0

ROGER!

TK 2 2 REPORTING REACHED DESTINATION 3133

TANK 13 REPORTS GIVING EFFECTIVE FIRE ON LN 2129
YOUR COMMAND:TK 13 MOVE TO 2128.

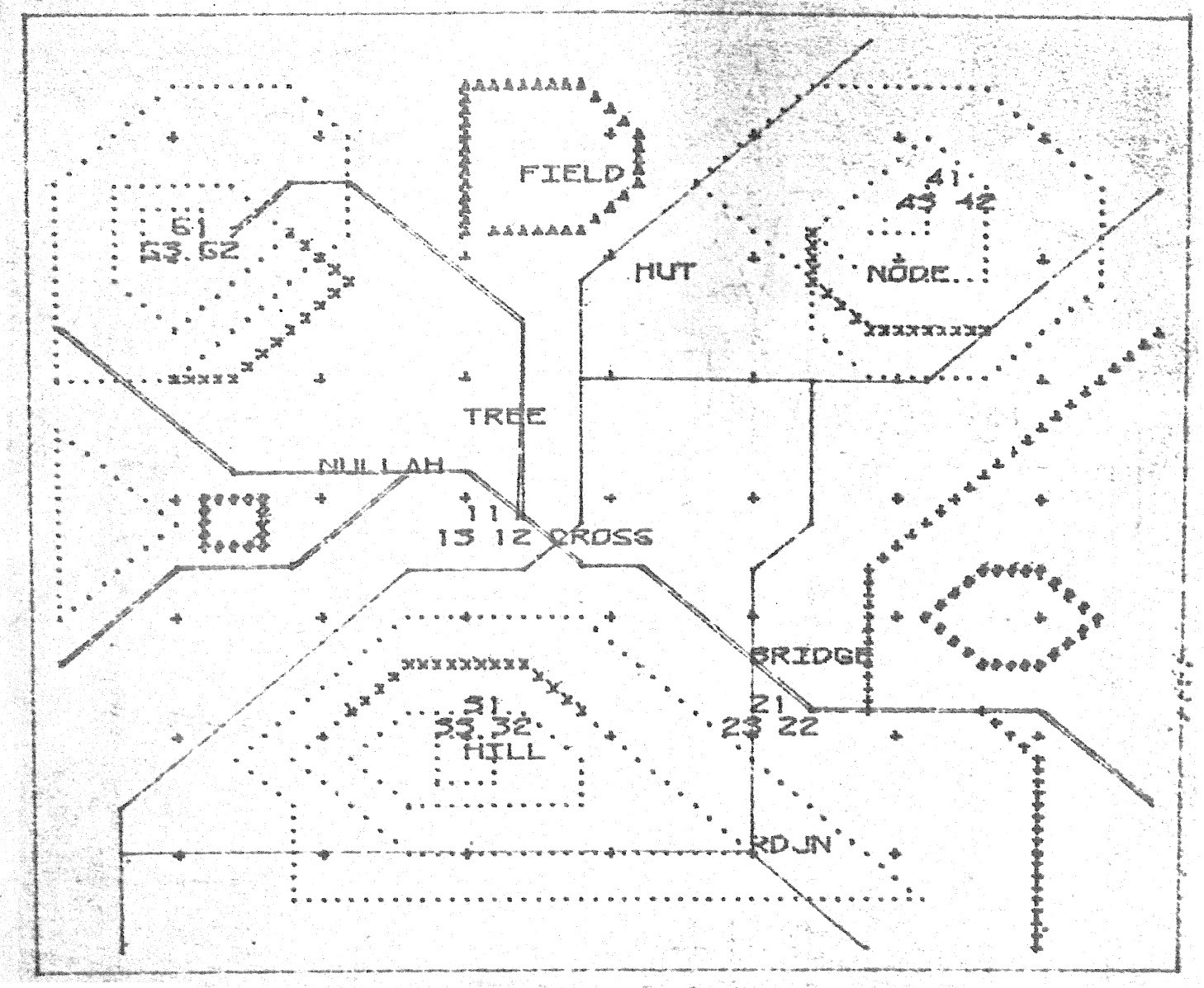
TIME:20: 8: 0

ROGER!

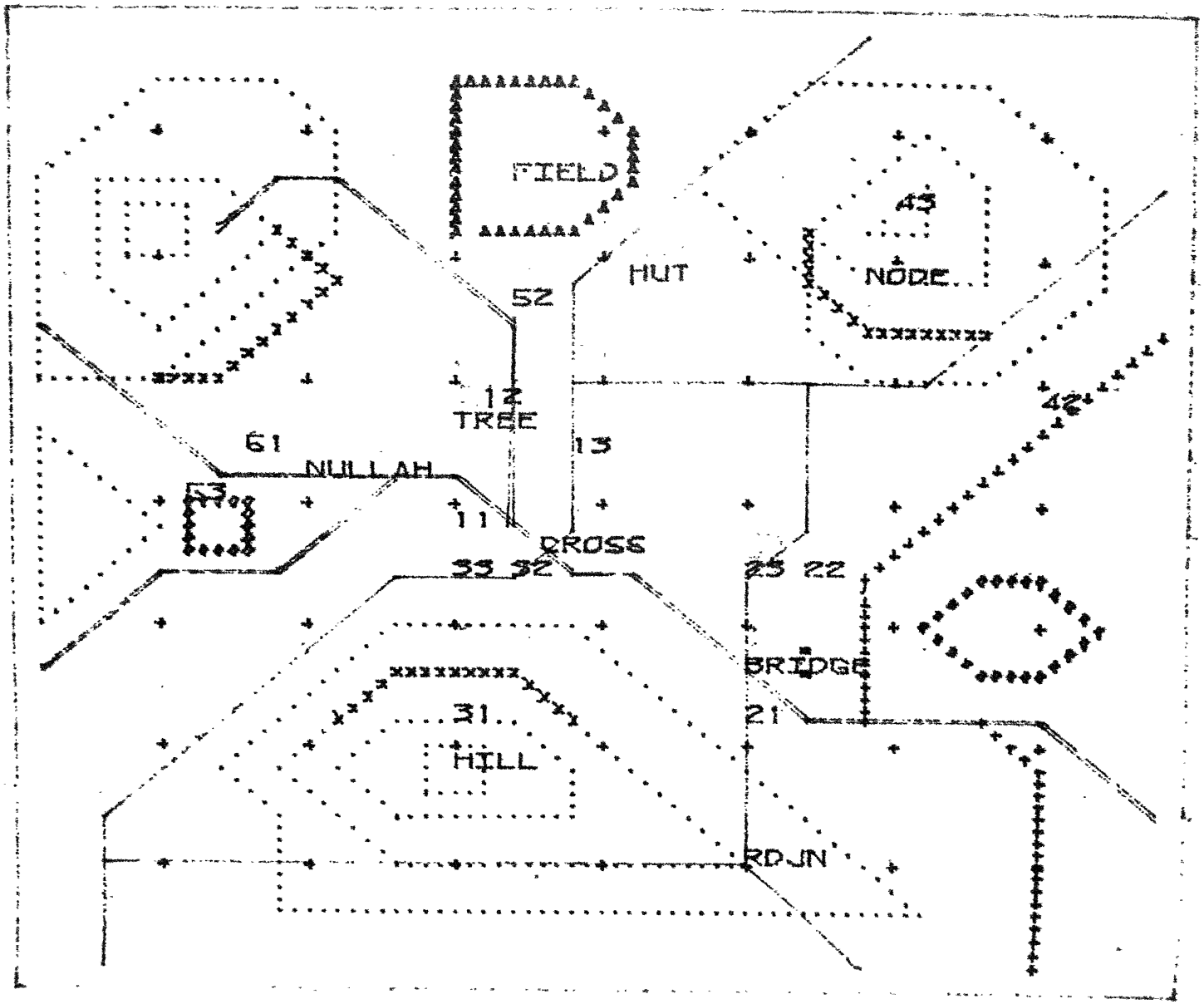
TK 1 3 REPORTING REACHED DESTINATION 2128
YOUR COMMAND:CONT.

TIME:20: 9: 0

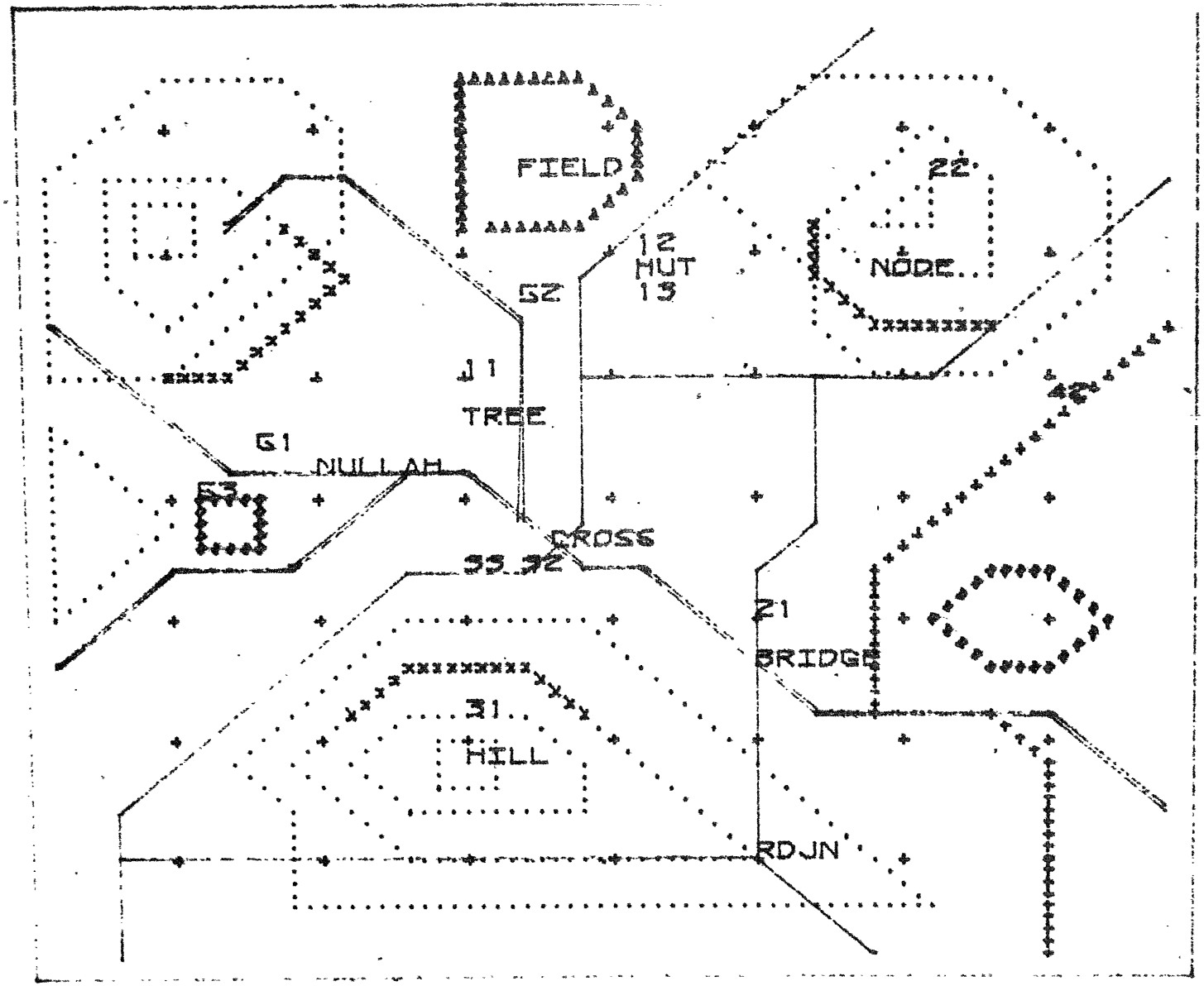
TK 1 2 REPORTING REACHED DESTINATION 2130
YOUR COMMAND:END.


FIELD
61
53.52
HUT
41
43 42
NODE
TREE
NULLAH
11
13 12 CROSS
BRIDGE
21
23 22
31
33.32
HILL
RDJN

MAP - 1


FIELD
HUT
52
43
NODE
12
TREE
61
NULLAH
13
42
11
CROSS
33
32
25
22
BRIDGE
31
HILL
21
RDJN

MAP-2


FIELD
12
HUT
13
22
NODE
52
11
TREE
42
G1
NULLAH
53
CROSS
33 32
21
BRIDGE
31
HILL
RDJN

MAP - 3

APPENDIX C

```
00010 {*****************************************************************\
00020   const
00030     LASTLIGHT=1050;FIRSTLIGHT=330;
00040   type
00050     AMN=record
00060      TYP1,TYP2,TYP3:integer
00070     end;
00080      CORD=record
00090      X,Y:integer
00100     end;
00110      STATREC=record
00120         BOGGED,MOVING,CANMOV,CANSHOOT,FIRING,CUPOLACLOSed,
00130         NUETRALISEd,SHOCKED:boolean
00140       end;
00150      AFVS=record
00160      TYPNO:1..5;MAPCORD:CORD;ORNTASN:0..360;AMNCAP:AMN;
00170      FUELCAP,ALT,LODTME:integer;STATUS:STATREC
00180     end;
00190      LMPTR=^LMTAB;
00200      LMTAB=record
00210       NAME:packed array [1..12]of char;
00220       GR:integer;
00230       RLINK,LLINK:LMPTR
00240      end;
00250      MPTR=^MOVETAB;
00260      MOVETAB=record
00270          ULINK,LLINK:MPTR;
00280          TANKNO:0..99;
00290          SOURCE,DEST,LOC:CORD;
00300          ROUTE:array[1..10]of CORD;
00310          STATE:boolean;
00320          OBJECTIVE,CURROBJECTive:integer
00330        end;
00340      AFVSCHAR=record
00350          LENGTH,WIDTH,HEIGHT,
00360          ARMRFRONT,ARMRSIDE,ARMRREST,
00370          SPEEDROAD,SPEEDCROSS,SPEEDWATER,
00380          STEPCLIMB,DITCHCLEAR,
00390          FUELCAP,FUELRES,MAXGRAD,
00400          DAYVISIBILITY,NIGHTVISIBility,
00410          LPKMCROSS,LPKMROAD:integer
00120        end;
00130     AFVSGUNCHar=record
00440      AMNCAPACITY,(*TOTAL AMMUNATION IT CAN CARRY*)
00450      MV,(*MUSSLE VELOCITY IN M/SEC*)
00460      RATEMOVE,(*RATE OF FIRING WHEN TANK ON THE MOVE*)
00470      RATEHALT,(*RATE OF FIRING WHEN THE TANK IS STATIONERY*)
00480      RANGEDIRECt,(*RANGE WHEN ENGAGING IN DIRECT FIRE*)
00490      RANGEINDIRect,(*RANGE WHEN ENGAGING INDIRECT FIRE*)
00500      ARM500,(*ARMOUR PEIRCING CAPABILITY WHEN DISTANCE OF
00510          TARGET IS 500 METERS*)
00520      ARM1000,(*ARMOUR PEIRCING CAPABILITY WHEN DISTANCE OF
00530          TARGET IS 1000 METEERS*)
00540      ARM2000,(*ARMOUR PEIRCING CAPABILITY WHEN THE DISTANCE OF
00550          TARGET IS>2000 METERS*)
00560      ELEVATION,(*MAXIMUM ELEVATION THAT IS POSSIBLE +/-1*)
```

```
00570              DEPRESSION:integer(*MAXIMUM DEPRESSION POSSIBLE +/-1*)
00580            end;
00590            LETTER=packed array[1..2]of char;
00600            NAMEWORD=packed array[1..14]of char;
00610            RANGE=0..8;
00620            GRIDPTR=^GRIDRECHAR;
00630            GRIDRECHAR=record
00640            NEXTPTR:GRIDPTR;
00650            VALUE1,VALUE2:integer
00660          end;
00670            GRIDRECORD=record
00680            ALTITUDE:integer;
00690            TYPOFGROUND:0..3;
00700            GROUNDPTR:GRIDPTR
00710          end;
00720            LETTERWORD=packed array[1..12]of char;
00730            FPTR=^FIRETAB;
00740            FIRETAB=record
00750                SOURCEIDX,DESTIDX:packed array[1..2]of integer;
00760                SOURCELOC,DESTLOC:COORD;
00770                BEGNUM,TIME:integer;
00780                MUZPROB:real;
00790                RATE,TYPOFAMN:0..15;
00800                ULINK,LLINK:FPTR
00810              end;
00820            {*****************************************************************\
00830        var
00840            LINEINPUT:packed array[1..80] of char;(*MAX LENGTH OF INPUT LINE*)
00850            WORD:packed array [1..12]of char;(*MAX SIZE OF ANY WORD*)
00860            TANKDAT:array[1..5,1..3]of AFVS;(*TANK IDENTIFICATION, FIRST SUFFIX=TANKNUMBER;
00870                                              SECOND SUFFIX=TROOP NUMBER*)
00880            K,LASTVAL,IK,JK,I,J,L,M:integer;
00890            NUMWORD:packed array[1..4]of 0..9;(*FOUR FIGURE GRID REFERENCE*)
00900            F,ISNUMBER,T:boolean;
00910            MR,MP,MQ,MLAST,MOVTABROOT:MPTR;(*POINTERS TO MOVE TABLE*)
00920            DEG:integer;
00930            ISLANDMARK:boolean;
00940            CH:char;
00950            LP,LQ,LR,LMROOT:LMPTR;(*POINTERS TO LANDMARKS TABLE*)
00960            TANKNUMBER:integer;
00970            TESTNUMBER:boolean;
00980            A,B:integer;
00990            DIST:integer;
01000            ANGLE:real;
01010            ISINMOVTAB,MAPREQUIRED,CROSSED:boolean;
01020            TIMESTEP,TIMESEC,TIMEMIN,TIMEHR:integer;(*USED FOR TIME STEP CALCULATION*)
01030            FREEZE:boolean;
01040            TANKCHAR:array[1..3]of AFVSCHAR;
01050            GUNCHAR:array[1..3]of AFVSGUNCHAR;
01060            GRX,GRY:LETTER;
01070            GRIDCHAR:array[0..41,0..41]of GRIDRECORD;TEMPPTR,PTR:GRIDPTR;
01080            MAPDAT,MAP,FIREDF:file of char;
01090            REACHEDDESTination:boolean;
01100            TYPOFTANK:1..4;
01110            ERRORPOINT:integer;
01120            FROOT,FLAST,FP,FQ,FR:FPTR;
```

```
01130            JOIN:boolean;
01140            MINEFIELDSTEP:integer;
01150            FIRSTREPORT:array[1..5,1..3] of boolean;
01160            HIT:array[1..5,1..3] of boolean;
01170            NAME:LANDWORD;{WAS:packed array[1..4]of char;
01180            %*************************************************************\
01190        procedure ERROR( L:integer);(*PRINTS THE TYPE OF ERROR WITH MESSAGES*)
01200        (* used for pointing out command errors during execution of programme *)
01210        %*************************************************************\
01220        var
01230            J:integer;
01240        begin
01250            WRITELN(TTY,LINEINPUT);for J:=1 to ERRORPOINT do WRITE(TTY,' ');
01260            WRITELN(TTY,'^');WRITELN(TTY);
01270            WRITE(TTY,'ERROR IN INPUT COMMAND-FOUND > ',WORD,' <WHEN EXPECTING ');
01280            WRITELN(TTY);
01290            case L of
01300            1:WRITELN(TTY,'STARTING SYMBOL TK');
01310            2:WRITELN(TTY,'TK  AT START OF A COMMAND');
01320            3:WRITELN(TTY,'STATUS,LOCATION,BALANCE,ENEMY,VISIBILITY');
01330            4:WRITELN(TTY,'<TANKNAME MUST END WITH ,/BLANK');
01340            5:WRITELN(TTY,'LESS THAN FOUR INTEGERS IN GRID REFERENCE');
01350            6:WRITELN(TTY,'INTEGERS IN GRID REFERENCE');
01360            7:WRITELN(TTY,'LESS THAN 80 CHAR IN ACOMMAND');
01370            8:WRITELN(TTY,'  AMMUNATION OR FUEL AFTER BALANCE');
01380            9:WRITELN(TTY,' <,> OR <TK> AFTER STATUS');
01390            10:WRITELN(TTY,' <LM> OR <TK> AFTER VISIBILITY');
01400            11:WRITELN(TTY,'UNIQUE LANDMARK OR GRIDREFERENCE');
01410            12:WRITELN(TTY,'LESS THAN 20 LANDMARKS OR INTEGERS IN GRID REF');
01420            14,18,21:WRITELN(TTY,'LANDMARK/GRIDREFERENCE');
01430            15:WRITELN(TTY,"."/"OF"/<DIST> AFTER MOVE');
01440            16,13:WRITELN(TTY,'<360 DEGREE IN DIRECTION');
01450            17:WRITELN(TTY,'MIR,MIRS,METER,KM,KMS');
01460            20:WRITELN(TTY,"ROUTE"/"." AFTER OBJ');
01470            22:WRITELN(TTY,'NO PREVIOUS COMMAND');
01480            23:WRITELN(TTY,'SMOKE/HE/AP ie type of ammunation');
01490            24:WRITELN(TTY,'TANKNAME/TANKNO');
01500            25:WRITELN(TTY,'TK/TANK after AT');
01510            26:WRITELN(TTY,'AT/GR after SHELL(S)');
01520            27:WRITELN(TTY,'SHELL(S) after typ of ammunation')
01530            end;
01540            BREAK
01550        end;
01560        %*************************************************************\
01570        procedure CHECKPTS;(*GENERAL GUIDELINES TO RUN THE PROGRAMME*)
01580        %*************************************************************\
01590        begin WRITE(TTY,'THIS IS A PROGRAMME EXERCISE FOR TANK BATTLE AT SQUADRON LEVEL.');
01600            WRITE(TTY,'YOU ARE REQUIRED TO TYPE IN COMMAND IN THE FORMAT SPECIFIED ');
01610            WRITE(TTY,'AND IT WILL BE EXECUTED.');
01620            WRITELN(TTY,'THE FOLLOWING POINTS NEED PARTICULAR ATTENTION:-');
01630            WRITELN(TTY,'1':10,' THE COMMAND MUST BE TYPED STARTING
01640        WITH <TK> OR OTHER COMMAND WORDS');
01650            WRITELN(TTY,'2':10,' THE COMMAND MUST END WITH <.>');
01660            WRITELN(TTY,'3':10,' IN CASE OF HELP TYPE <HELPCOMMAND> OR <HELP>');
01670            WRITELN(TTY,'3':10,' TO TERMINATE THE PROGRAMME TYPE <END.>');
01680            WRITELN(TTY,'4':10,' AFTER YOU HAVE WRITTEN YOUR
```

```
01690       COMMAND GIVE CARRAIGE RETURN');
01700             WRITELN(TTY,'S':10,' IF YOU WANT THESE COMMANDS TO BE
01710       TYPED THEN TYPE COMMAND <CR>KEYS.>');
01720             WRITELN(TTY,'READY TO START NOW');
01730             BREAK
01740         end;
01750             %*************************************************************\
01760         procedure MSG(M:integer);(*PRINTS MESSAGES*)
01770         (* this is primarily used to print out messages originating from the programme *)
01780         %*************************************************************\
01790         begin
01800             case M of
01810             1:WRITELN(TTY,'ENIGMA TANK ALREADY ON MOVE');
01820             2:WRITELN(TTY,'EXPECTING A D/R/C/W HENCE "W" ASSUMED');
01830             3:WRITELN(TTY,'MORE THAN 12 BEING IGNORED');
01840             4:WRITELN(TTY,'TANK BOGGED CANNOT MOVE');
01850             5:WRITELN(TTY,'CAN'T DO THE MOVE');
01860             6:WRITELN(TTY,'TANK GUN FIRING ON THE TARGET');
01870             7:WRITELN(TTY,'TANK GUN OUT OF ACTION - CANNOT FIRE');
01880             8:WRITE(TTY,'TANK CANNOT MOVE DUE TO ');
01890             9:WRITELN(TTY,'HIT BY ENEMY FIRE');
01900             10:WRITELN(TTY,'MECHANICAL FAILURE');
01910             11:WRITELN(TTY,'DIRECTION WRONG-GOING OUT OF MAP AREA');
01920             12:WRITE(TTY,'TANK',IK:2,JK:2,' REPORTING > ');
01930             13:WRITELN(TTY,'TANK HAS RAN OUT OF FUEL');
01940             14:WRITELN(TTY,'TANK RUNNING ON RESERVE');
01950             15:WRITELN(TTY,'WATER TOO DEEP - TANK IS BOGGED');
01960             16:WRITELN(TTY,'CANNOT NEGOTIATE THE CANAL - SIDES ARE VERY STEEP');
01970             17:WRITELN(TTY,'SHOULD HAVE GIVEN DIRECTION ie BACK/LEFT etc');
01980             18:WRITELN(TTY,'SORRY! NOT ABLE TO SHOOT-IK',IK:1,JK:1);
01990             19:WRITELN(TTY,'CANNOT SEE AT THAT DISTANCE AT NIGHT');
02000             20:WRITELN(TTY,'CANNOT SEE THAT DISTANCE IN DAY');
02010             21:WRITELN(TTY,'LOTS OF COVER AT THAT LOCATION')
02020             end;
02030             BREAK
02040         end;
02050             %*************************************************************\
02060         procedure HELPREPORT;(*PRINTS COMMANDS AVAILABLE IN 'REPORT PROCEDURE'*)
02070         (* lists all possible commands available in REPORT command - used in the case of HEL
02080         %*************************************************************\
02090         begin
02100             WRITELN(TTY,' the following commands are possible in report');
02110             WRITELN(TTY,'command with appropriate options');
02120             WRITELN(TTY,'  ':5,'(tk <tankname>) report location.');
02130             WRITELN(TTY,'  ':27,'status tk <tankname>/s  loc <grid ref>/a  .');
02140             WRITELN(TTY,'  ':27,'balance ammunation/fuel he/smoke/ap/all  .');
02150             WRITELN(TTY,'  ':27,'enemy .');
02160             WRITELN(TTY,'  ':27,'visibility tk <tankname>/loc <grid ref> .');
02170             WRITELN(TTY,'  ':10,' note:<tankname>::=tank no followed by troop no:');
02180             WRITELN(TTY,'  <grid REF>::=six figure grid ref') ;
02190             BREAK
02200         end;
02210             %*************************************************************\
02220         procedure COMMANDHELP;
02230         %*************************************************************\
02240         begin
```

```
02250          WRITELN(TTY,'THE FOLLOWING COMMANDS ARE POSSIBLE AS "COMMANDS"');
02260          WRITELN(TTY,'  :S,'OK <no> <firepart>/<movepart>/<reportpart>.');
02270          WRITELN(TTY,'  :S,'HELP/HELPCOMND.');
02280          WRITELN(TTY,'  :S,'LANDMARK <gives the listing of defined landmarks>');
02290          WRITELN(TTY,'  :S,'REPORT/STATAB <gives the listing of tanks');
02300          WRITELN(TTY,'  DO the move with details>' );
02310          WRITELN(TTY,'  :S,'FREEZE <freezes all ACTION part for the duration>');
02320          WRITELN(TTY,'  :S,'START/CONTINUE <continues the ACTION part after freeze>');
02330          WRITELN(TTY,'  :S,'END. <stops the programme completely>');
02340      end;
02350        %*****************************************************************\
02360      procedure HELPMOVE;(*LISTS COMMANDS AVAILABLE IN 'MOVE' PROCEDURE*)
02370      (* lists all possible commands available in MOVE command - used in the case of HELP
02380      %*****************************************************************\
02390      begin
02400          WRITE(TTY,'THE FOLLOWING COMMANDS ARE POSSIBLE IN "MOVE"');
02410          WRITELN(TTY,'  COMMANDS WITH APPROPRIATE OPTIONS:');
02420          WRITELN(TTY,'(OK <TANKNAME>)MOVE.');
02430          WRITELN(TTY,'  :10,'TO LANDMARK.');
02440          WRITELN(TTY,'  :20,'ROUTE <LANDMARKS>.');
02450          WRITELN(TTY,'  :10,'<DISTANCE> MTR/METER <DIRECTION>.');
02460          WRITELN(TTY,'  :50,'OF <LANDMARK>.');
02470          WRITELN(TTY,'  :50,'ROUTE <LANDMARKS>.');
02480          WRITELN(TTY,'DIRECT:cardinal-direct::=north/n,north-east/ne,etc.');
02490          WRITELN(TTY,'  :1,'direction::=clock-direction/cardinal-direction/degree');
02500          WRITELN(TTY,'  :1,'clock-direct::=12 O"clock,1 O"clock etc.');
02510          WRITELN(TTY,'  :1,'landmark::=<defined landmark name>');
02520          WRITELN(TTY,'  :1,'landmarks::=<landmark>,<landmark>.');
02530      end;
02540        %*****************************************************************\
02550      procedure HELPFIRE;(* PRINTS COMMANDS THAT ARE AVAILABLE IN FIRE COMMAND*)
02560      %*****************************************************************\
02570      begin
02580          WRITELN(TTY,'the following commands are possible in FIRE commands');
02590          WRITELN(TTY,'  :S,'OK <no> FIRE/SHOOT <no> MR/AP/SMOKE SHELL(S) AT OK <no>.');
02600          WRITELN(TTY,'  :50,'<direction>.');
02610          WRITELN(TTY,'  :50,'OK <gridref>.');
02620          WRITELN(TTY,'  :53,'OK <gridref>.');
02630          WRITELN(TTY,'  :13,'STOP FIRING');
02640          WRITELN(TTY,'NOTE->direction::=north/front/right/left etc');
02650      end;
02660        %*****************************************************************\
02670      procedure READCH;(*READS A CHARACTER FROM THE INPUT COMMAND*)
02680      (* reads the first char from the input command and
02690    stores in CH. used for yes/no type of dialogue *)
02700        %*****************************************************************\
02710      var
02720          ALP:char;
02730      begin
02740          repeat READ(TTY,CH)
02750          until CH#' ';
02760          while not EOLN(TTY) do READ(TTY,ALP);
02770      end;
02780        %*****************************************************************\
02790      procedure TYPOFHLP;(*DECIDES TYPE OF HELP REQUIRED*)
02800      (* in case of HELP decides the type of HELP required *)
```

```
02810         %*************************************************************\
02820         begin
02830              READCH:
02840              case CH of
02850              'R':HELPREPORT;
02860              'M':HELPMOVE;
02870              'F':HELPFIRE;'C':COMMANDHELP;
02880              'E':;
02890              others:
02900                begin WRITELN(TTY,'your>',CH,'<not recognised please retype:');
02910                      BREAK;GOTO READCH
02920                end
02930              end;
02940         end;
02950              %*************************************************************\
02960         procedure HELPCOMMAND;(*PRINTS 'HELP' IN COMMAND PROCEDURE*)
02970         %*************************************************************\
02980         begin
02990              WRITELN(TTY,'yes! this is helper for your commands,
03000         please type char for the help required:-');
03010              WRITELN(TTY,' E ',' for return to execution :E ');
03020              WRITELN(TTY,' R ',' for report commands :R ');
03030              WRITELN(TTY,' M ',' for move commands :M ');
03040              WRITELN(TTY,' F ',' for fire commands :F ');
03050              WRITELN(TTY,' C ',' for COMMAND commands :C ');
03060              WRITE(TTY,' ',' help required for:');BREAK;TYPEHELP
03070         end;
03080              %*************************************************************\
03090         procedure READLINE;(*READS IN ONE LINE INPUT AND STORES IN 80 LINE BUFFER*)
03100         (* reads upto 80 chars from the tty to process the command.
03110              it is terminated by "."or EOLN or >80 char *)
03120         %*************************************************************\
03130         var
03140              IL:integer;ALP:char;
03150         begin  for IL:=1 to 80 do LINEINPUT[IL]:=' ';
03160              ERRORPOINT:=1;
03170              IL:=1;
03180              repeat READ(TTY,ALP);LINEINPUT[IL]:=ALP;
03190              IL:=IL+1
03200              until ((ALP='.')or(IL>80)or(EOLN(TTY)));
03210              if ALP#'.' then
03220                begin WRITE(TTY,'MISSING POINT AT END PLEASE RETYPE:');
03230              BREAK;READLINE
03240                end
03250              else
03260              I:=1
03270         end;
03280              %*************************************************************\
03290         procedure READWORD;(*READS ONE WORD FROM THE INPUT COMMAND*)
03300         (* takes the next word from the input command for processing. input to
03310              this from the buffer LINEINPUT *)
03320         (* delimiters between words are "b","," and for termination
03330              in addition a "." is included. *)
03340         (* it also stores the last word location in ERRORPOINT for error messages *)
03350         %*************************************************************\
03360         var
```

```
03370           J:integer;
03380       begin for J:=1 to 12 do WORD[J]:=' ';
03390           J:=1;while((LINEINPUT[I]=' ')or(LINEINPUT[I]=',')) do I:=I+1;
03400           ERRORPOINT:=I;
03410           if I<80 then
03420             begin
03430           repeat
03440           if J<13 then WORD[J]:=LINEINPUT[I];
03450           J:=J+1;I:=I+1
03460           until
03470           ((LINEINPUT[I]=' ')or(LINEINPUT[I]='(')or(LINEINPUT[I]=')')
03480           or(LINEINPUT[I]=',')or(WORD[J-1]='(')
03490           or(WORD[J-1]=')')or(LINEINPUT[I]=','));
03500           if J>12 then
03510           ERR(3)
03520             end
03530           else ERROR(7)
03540       end;
03550           %*************************************************************************\
03560       procedure READIRAW;(*READS DIRECTLY INPUT UPTO 12 CHARACTERS*)
03570       (* reads directly 12 char from tty and stores them in WORD.
03580       used to interactive communication   *)
03590       %*************************************************************************\
03600       var
03610           INCH:char;I:integer;
03620       begin
03630           for I:=1 to 12 do WORD[I]:=' ';
03640           INCH:=' ';I:=1;
03650           while INCH=' ' do READ(TTY,INCH);
03660           while ((not EOLN(TTY))and(I<13)and(INCH#',')) do
03670             begin
03680           WORD[I]:=INCH;I:=I+1;READ(TTY,INCH);
03690             end;
03700           if I>12 then ERR(3)
03710           else
03720           if EOLN(TTY) then WORD[I]:=INCH
03730       end;
03740           %*************************************************************************\
03750       procedure NUMBER;(*CONVERTS CHAR NUMBER TO INTEGER*)
03760       (* this procedure checks if the current word in the buffer WORD is an integer.
03770        if testing is set by POSTNUMBER then   sets this *)
03780       (* to TRUE if it is a NUMBER and converts it to integer return in INTVAL *)
03790       %*************************************************************************\
03800       var
03810           M,N:integer;CH:char;
03820       begin for M:=1 to 4 do NUMWORD[M]:=0;
03830           M:=1;INTVAL:=0;ISNUMBER:=true;
03840           loop
03850           CH:=WORD[M];
03860           exit if ((CH=' ')or(M>4)or(ORD(CH)>ORD('9'))or(ORD(CH)<ORD('0')));
03870           N:=ORD(CH)-ORD('0');NUMWORD[M]:=N;
03880           M:=M+1;INTVAL:=INTVAL*10+N
03890           end;
03900           if ((CH#' ')and(M>4)) then ERROR(5);
03910           if (CH#' ') then
03920             begin
```

```
03930          ISNUMBER:=false;
03940          if TESTNUMBER then
03950            begin
03960            ERROR(6);WRITE(TTY,'WOULD YOU LIKE TO SUBSTITUTE THIS NUMBER ?:');
03970            BREAK;READCH;
03980            if CH='Y' then
03990              begin
04000                WRITE(TTY,'YOUR NEW NUMBER:');BREAK;READNUMBER(NUMBER
04010              end;
04020            end;
04030          TESTNUMBER:=false
04040      end;
04050          {*****************************************************************\
04060      procedure SKIP(P:integer);(*SKIPS 'P' LINES WHILE OUTPUTTING*)
04070      (* used to skip lines on tty for clearity of presentation *)
04080      {*****************************************************************}
04090      var
04100          I:integer;
04110      begin for I:=1 to P do WRITELN(TTY)
04120      end;
04130          {*****************************************************************\
04140      function YES:boolean;(*CHECKS ANS*)
04150      {*****************************************************************}
04160      begin
04170          READCH;
04180          if CH='Y' then YES:=true
04190          else YES:=false
04200      end;
04210          {*****************************************************************\
04220      function REDUCE(X:integer):real;(*reduces the scale from integer to real for graphic
04230      {*****************************************************************}
04240      begin
04250          REDUCE:=X/40
04260      end;
04270          {*****************************************************************\
04280      procedure CONVERT(INT:integer;var A,B:integer);
04290      (*SPLITS THE GRIDREF TO NORTHINGS AND EASTINGS*)
04300      (* INT is the input value and A,B give the northings and eastings of the location *
04310      {*****************************************************************}
04320      begin
04330          B:=INT mod 100;
04340          A:=INT div 100
04350      end;
04360          {*****************************************************************\
04370      procedure CHECKLANDmark(var CH:integer);
04380          forward;
04390          {*****************************************************************\
04400      procedure CALCULATEDist(X1,Y1,X2,Y2:integer;var DISTANCE:integer);
04410      (* calculates the distance between gridref x1,y1 and x2,y2 and outputs it in DISTANC
04420      {*****************************************************************}
04430      var
04440          TEMP:real;
04450      begin
04460          TEMP:=((X1-X2)*(X1-X2) + ((Y1-Y2)*(Y1-Y2)));
04470          DISTANCE:=TRUNC(SQRT(TEMP)+0.5)
04480
```

```
04490          end;
04500              %*************************************************************\
04510          procedure PRTLMTAB(P:LMPTR);(*PRINTS TABLE OF LANDMARK OFFSETS)*)
04520          (* it does the inorder printing of LANDMARKS defined by the user and stored in
04530           and accessed through LMPTR *)
04540          %*************************************************************\
04550          begin
04560              if P<>LMROOT then PRTLMTAB(P^.LLINK);
04570              if (P^.LLINK=nil) then WRITELN(TTY,'LANDMARK NAME':25,'GRID REF':20);
04580              WRITELN(TTY);
04590              if P<>LMROOT then
04600              WRITELN(TTY,P^.NAME:25,P^.GR:20);
04610              if (P^.RLINK=nil) then PRLMTAB(P^.RLINK)
04620          end;
04630              %*************************************************************\
04640          procedure UPDATE;(*UPDATES/ADDS TO LANDMARK TABLE*)
04650          %*************************************************************\
04660          var
04670              R:boolean;I:integer;
04680          begin
04690              WRITE(TTY,'DO YOU WANT TO DEFINE LANDMARKS ?');BREAK;
04700              READCH(CH);
04710              if CH='Y' then
04720                begin
04730                WRITELN(TTY,'TYPE IN LANDMARKS IN THE FOLLOWING FORMAT (<LM>,<GR>.)');
04740                WRITELN(TTY,'WHEN YOU HAVE FINISHED then type [FINISH.]');
04750                loop
04760                I:=I+1;WRITE(TTY,'INPUT:');BREAK;READLINE;READWORD;
04770                if WORD='FINISH      ' then
04780                  begin NEW(LQ);LQ^.NAME:=WORD;READWORD;NUMBER
04790                  end;
04800                R:=true;
04810                exit if ((WORD='FINISH      ')or(I>20)or(ISNUMBER=false));
04820                LQ^.GR:=NUMVAL;LQ^.LLINK:=nil;LQ^.RLINK:=nil;
04830                if LMROOT=nil then
04840                  begin
04850                    LMROOT:=LQ;LMROOT^.RLINK:=LQ
04860                  end
04870                else
04880                  begin
04890                    LP:=LMROOT^.RLINK;
04900                    while ((LP<>nil)and(R=true)and(LP^.GR<>LQ^.GR)) do
04910                  begin
04920                    LR:=LP;
04930                    if LQ^.NAME=LP^.NAME then R:=false
04940                    else
04950                      begin
04960                    if LQ^.NAME>LP^.NAME then LP:=LP^.RLINK
04970                    else LP:=LP^.LLINK
04980                      end
04990                  end;
05000                  if LP=nil then
05010                  begin
05020                    if LQ^.NAME>LR^.NAME then LR^.RLINK:=LQ
05030                    else LR^.LLINK:=LQ
05040                  end
```

```
05050                 else ERROR(11)
05060               end
05070             end;
05080             if (WORD='FINISH        ') then ERROR(12)
05090               end;
05100             WRITE(TTY,'DO YOU WANT THE TABLE OF LANDMARKS ?');
05110             BREAK;
05120             READCH;
05130             if CH='Y' then
05140             PRINTTAB(LANDP);
05150         end;
05160             {*****************************************************************\
05170         procedure READTIMESTEP;(*READS IN TIMESTEP TO BE USED*)
05180         {*****************************************************************\
05190         var
05200             A,B:integer;
05210         begin
05220             PROMPT:=false;
05230             WRITE(TTY,'DO YOU WANT TO CHANGE TIMESTEP? [BY  DEFAULT IT IS TAKEN AS 1 MIN] :'
05240             BREAK;READCH;
05250             if CH='Y' then
05260               begin
05270               WRITE(TTY,'PLEASE TYPE TIMESTEP IN FORMAT [<TIME> HR/MIN/SEC .] :');BREAK;
05280               READLINE;READWORD;NUMBER;
05290               if ISNUMBER then
05300                 begin
05310                 READWORD;
05320                 if WORD='MIN           ' then TIMESTEP:=INTVAL*60
05330                 else
05340                 if WORD='HR            ' then TIMESTEP:=INTVAL*3600
05350                 else
05360                 if WORD='SEC           ' then TIMESTEP:=INTVAL
05370                   end
05380                   end
05390             else TIMESTEP:=60;
05400             WRITE(TTY,'GIVE TIME OF START OF EXERCISE IN (HR,MIN) 4 FIGURES:');BREAK;
05410             READINPUT;NUMBER;CONVERT(INTVAL,A,B);
05420             TIMESEC:=A*3600+B*60;TIMEHR:=0;
05430             MINTIMESTEP:=TIMESTEP div 60;
05440         end;
05450             {*****************************************************************\
05460         procedure INITGRID;(*initialises the grid points to plain ground and rest to 0 *)
05470         {*****************************************************************\
05480         var
05490             I,J:integer;
05500         begin
05510             for I:=1 to 40 do
05520             for J:=1 to 40 do
05530             with GRIDCHAR[I,J] do
05540               begin
05550                 ALTITUDE:=0;
05560                 TYPOFGROUND:=3;
05570                 GROUNDPTR:=nil
05580               end;
05590         end;
05600             {*****************************************************************\
```

```
05610       procedure CALCNEXTGRIDPOINT(var X,Y:integer;DIRECTION:RANGE);
05620       (* given the DIRECTION and the present location in X,Y
05630       it gives the next gridpoint in X,Y *)
05640       {*****************************************************************\
05650       begin
05660           case DIRECTION of
05670           0:Y:=Y+1;
05680           1:
05690             begin X:=X+1;
05700               Y:=Y+1;
05710             end;
05720           2:
05730           X:=X+1;
05740           3:
05750             begin
05760                 X:=X+1;
05770                 Y:=Y-1;
05780             end;
05790           4:
05800           Y:=Y-1;
05810           5:
05820             begin X:=X-1;
05830               Y:=Y-1
05840             end;
05850           6:
05860           X:=X-1;
05870           7:
05880             begin
05890                 X:=X-1;
05900                 Y:=Y+1
05910             end
05920           end;
05930       end;
05940       {*****************************************************************\
05950       procedure INITIALISE;
05960       (* initialises the details for the complete programme *)
05970       {*****************************************************************\
05980       begin
05990           RESET(MAPDAT);
06000           REWRITE(FIREDI);
06010           PTTOFV(1);
06020           CHECKPTS;
06030           NEW(LMROOT);
06040           LMROOT^.RLINK:=nil;LMROOT^.LLINK:=nil;LP:=LMROOT;LMROOT^.MARF:=
06050           RESNUMBER:=true;
06060           NEW(MOVTABROOT);MOVTABROOT^.LLINK:=nil;MP:=MOVTABROOT;MBASE:=MP;
06070           MP^.LLINK:=nil;ISGAMMARK:=false;
06080           NEW(FROOT);FBASE:=FROOT;FROOT^.LLINK:=nil;FROOT^.JLINK:=nil;
06090           MAPREQUIRED:=false;
06100           with TANKCHAR[1] do
06110             begin
06120           LENGTH:=900;(*CM*)
06130           WIDTH:=327;(*CM*)
06140           HEIGHT:=235;(*CM*)
06150           ARMRFRONT:=200;(*MM*)
06160           ARMRSIDE:=160;(*MM*)
```

```
        ARMRREST:=60;(*MM*)
        SPEEDROAD:=50;(*KMPH*)
        SPEEDCROSS:=30;(*KMPH*)
        SPEED AIFR:=9;(*KMPH*)
        STEPCLIMB:=8;(*IN 0.1 METER *)
        DITCHCLEAR:=27;(*IN 0.1 METER *)
        FUELCAP:=620;(*LITRES*)
        FUELRES:=300;(*LITRES*)
        MAXGRAD:=32;(*DEGREES*)
        OPKMCROSS:=3;(*LITRES PER KM*)
        DAYVISIBILITY:=8;
        NIGHTVISIBILITY:=3;
        LPKMROAD:=2;(*LITRES PER KM ON ROAD *)
          end;
        with GUNCHARIII do
          begin
        AMNCAPACITY:=43;(*NOS*)
        MV:=900;(* M/SEC *)
        RATEMOVE:=9;(* ROUNDS PER MINUTE*)
        RATEHALT:=7;(* ROUNDS PER MINUTE*)
        RANGEDIRECT:=7000;(*METERS*)
        RANGEINDIRect:=14600;(*METERS*)
        ARM500:=200;(*MM*)
        ARM1000:=185;(*MM*)
        ARM2000:=155;(*MM*)
        ELEVATION:=19;(*DEGREES*)
        DEPRESSION:=4;(*DEGREES*)
          end
    end;
        %***************************************************************\
    procedure INITTANK;
    (* initialises the various location and other details regarding the type of tank
     used in the programme. the details are taken interactively from the user*)
    %****************************************************************\
    var
        I,ADD,J,X,Y,AMN1,AMN2,AMN3:integer;
    begin
        repeat
        WRITELN(TTY,'PLEASE GIVE THE TYPE OF TANK USED - ');
        WRITELN(TTY,'if it is VIJAYANTA then type "V"');
        WRITELN(TTY,'if it is T-55 then type "T"');
        WRITE(TTY,'type of tank :');BREAK;
        READCH;
        case CH of
        'V':
          begin
              TYPOFTANK:=1;
              AMN1:=10;AMN2:=20;AMN3:=13
          end;
        'T':
          begin
              TYPOFTANK:=1;
              AMN1:=10;AMN2:=20;AMN3:=13
          end;
        others:WRITELN(TTY,'YOUR ',CH,' NOT RECOGNISED PLEASE RETYPE -')
        end;
```

```
06730           until ((CH='Y')or(CH='F'));
06740           WRITELN(TTY,'PROGRAM GIVE THE LOCATION OF VARIOUS GROUPS OF YOUR SQUADRON');
06750           WRITELN(TTY,'YOU ARE REQUIRED TO GIVE ONLY FOUR FIGURE GRID REFERENCE');
06760           for J:=1 to 5 do
06770             begin
06780           WRITE(TTY,'GROUP NO',J:3,':');BREAK;
06790           READLN(FOT);CONVERT(INTVAL,X,Y);
06800           ADD:=1;
06810           for I:=1 to 3 do
06820           with TANKCAR[I,J] do
06830             begin
06840                 TYPOFTANK:=TYPNO;
06850                 FUELCAP:=TANKCHAR[TYPOFTANK].FUELRES + TANKCHAR[TYPOFTANK].
06860                 FUELCAP;
06870                 LOADING:=1;
06880                 if I=1 then
06890           begin MAPCORD.X:=X;MAPCORD.Y:=Y;
06900           end
06910                 else
06920           begin
06930                 MAPCORD.X:=X+ADD;MAPCORD.Y:=Y-1;ADD:=ADD*(-1);
06940           end;
06950                 AMNCAP.TYP1:=AMN1;
06960                 AMNCAP.TYP2:=AMN2;AMNCAP.TYP3:=AMN3;ALT:=GRIDCHAR[X,Y].ALTITUDE;
06970
06980                 with STATUS do
06990           begin
07000                 BOGGED:=false;MOVING:=false;
07010                 FIRING:=false;CUPOLACLOSed:=true;
07020                 MENTALISED:=false;CANMOV:=true;
07030                 CANSHOOT:=true;SHOCKED:=false
07040           end
07050           end
07060           end;
07070       %*****************************************************************\
07080       procedure CONVERTGRid(var GRX,GRY:LETTER;X,Y:integer);
07090       (* converts the integer cordinates "X","Y" to grid ref X,Y
07100          and outputs  GRX,GRY as chars *)
07110       (* these are used for storage and in printing out messages *)
07120       %*****************************************************************\
07130       var
07140           I,J,X1,Y1:integer;
07150       begin
07160           X1:=X;Y1:=Y;
07170           for I:=2 downto 1 do
07180             begin
07190           J:=(X1)mod(10);X1:=(X1)div(10);GRX[I]:=CHR(J+ORD('0'));
07200           J:=(Y1)mod(10);Y1:=(Y1)div(10);GRY[I]:=CHR(J+ORD('0'));
07210             end
07220       end;
07230       %*****************************************************************\
07240       procedure UPDATETIME;(*UPDATES THE TIME ELAPSED IN HR,MIN,SECS)
07250       %*****************************************************************\
07260       begin
07270           if not FREEZE
07280
```

```
07290             then
07300               begin
07310               TIMESEC:=TIMESEC+TIMESTEP;
07320               if TIMESEC>=60 then
07330                 begin
07340                 TIMEMIN:=TIMEMIN+(TIMESEC)div(60);TIMESEC:=(TIMESEC)mod(60)
07350                 end;
07360               if TIMEMIN>=60 then
07370                 begin
07380                 TIMEHR:=TIMEHR+(TIMEMIN )div (60);
07390                 TIMEMIN:=TIMEMIN mod 60;
07400                 end;
07410               if TIMEHR>=24 then TIMEHR:=TIMEHR-24;
07420                 end;
07430               WRITELN(TRY,'TIME:':50,TIMEHR:2,':',TIMEMIN:2,':',TIMESEC:2)
07440           end;
07450             {*****************************************************************\
07460           procedure MAINDIRECTION(DIR:RANGE;var DIRWORD:LETTERWORD);
07470           (* converts DIRECTION to the cardinal direction .
07480           used for outpution -like flow of river *)
07490             {*****************************************************************\
07500           begin
07510               case DIR of
07520               0:DIRWORD:='NORTH      ';1:DIRWORD:='NORTH EAST  ';
07530               2:DIRWORD:='EAST       ';3:DIRWORD:='SOUTH EAST  ';
07540               4:DIRWORD:='SOUTH      ';5:DIRWORD:='SOUTH WEST  ';
07550               6:DIRWORD:='WEST       ';7:DIRWORD:='NORTH WEST  ';
07560               others:
07570               end
07580           end;
07590             {*****************************************************************\
07600           procedure REMOVESHOCK;
07610           (* resets the tank to resume its activity after it has been hit
07620         and SHOCKED out of action *)
07630           var
07640               I,J:integer;
07650           begin
07660               for I:=1 to 5 do
07670               for J:=1 to 3 do
07680               with TANKDATA[I,J].STATUS do
07690                   if SHOCKED then SHOCKED:=false
07700           end;
07710             {*****************************************************************\
07720           procedure GETDIRECTION(X1,X2,Y1,Y2:integer;var DIRECTION:integer);
07730           {*****************************************************************\
07740           var
07750               XDISP,YDISP:integer;ANGLE:real;
07760               {*************************************************\
07770           procedure CALLANGLE(BASE:integer);
07780           {*************************************************\
07790                 begin
07800               if ANGLE>=2.35 then DIRECTION:=0
07810               else
07820               if ANGLE >= 0.42 then DIRECTION:=ABS(1-BASE)
07830               else DIRECTION:=ABS(2-BASE)
07840                 end;
```

```
07850          %*****************************************************\
07860      procedure CALLANGLEY(BASE:integer);
07870      %*****************************************************\
07880          begin
07890          if ANGLE <= 0.42 then DIRECTION:=ABS(BASE-2)
07900          else
07910          if ANGLE <= 2.35 then DIRECTION:=ABS(BASE-3)
07920          else DIRECTION:=4
07930          end;
07940      begin
07950          XDISP:=X2-X1;
07960          YDISP:=Y2-Y1;
07970          if XDISP=0 then
07980            begin
07990          if YDISP > 0 then DIRECTION:=0
08000          else DIRECTION:=4
08010            end
08020          else
08030            begin
08040          ANGLE:=ABS(YDISP/XDISP);
08050          if XDISP >0 then
08060            begin
08070          if YDISP>0 then CALLANGLEX(0)
08080          else CALLANGLEY(0)
08090            end
08100          else
08110          if YDISP>0 then CALLANGLEX(8)
08120          else CALLANGLEY(8)
08130            end
08140      end;
08150          %*****************************************************\
08160      procedure FILLUPDGRaphy;
08170      (* takes the details described earlier from file MAPDAT and stores it for use*)
08180      (* when this is completed a note to this effect is given on TTY *)
08190      %*****************************************************\
08200      type
08210          SPLITARRAY=packed array[1..2]of 0..99;
08220      var
08230          WIDTH,HEIGHT,DEPTH,I,DEGREE:integer;
08240          CUR1,CUR2:integer;
08250          FIXEDWORD:packed array[1..10]of char;
08260          DIRECTION:0..8;NEXTPOINTER,LASTPOINTER,CURRENTPOINTER:SPLITARRAY;
08270          FIRSTPOINT,ERROR:boolean;
08280          %*****************************************************\
08290      procedure GETPTR;
08300      %*****************************************************\
08310          begin
08320          NEW(PTR);PTR^.NEXTPTR:=nil;
08330          PTR^.VALUE1:=0;
08340          PTR^.VALUE2:=0
08350            end;
08360          %*****************************************************\
08370      procedure  FILLGRIDChar(P1,P2:integer);
08380      %*****************************************************\
08390          begin
08400          GETPTR;
```

```
08410           GRIDCHAR[P1,P2].GROUNDPTR:=PTR;
08420           GRIDCHAR[P1,P2].TYPOFGROUND:=1
08430             end;
08440           %*************************************************************\
08450         procedure COMBINETYPofground(P1,P2:integer);
08460         (*checks for ground when it has 2 or more characteristics at one grid point *)
08470         %*************************************************************\
08480             var
08490           TEMPGROUND:1..9;
08500             begin
08510           TEMPGROUND:=GRIDCHAR[P1,P2].TYPOFGROUND;
08520           GRIDCHAR[P1,P2].TYPOFGROUND:=7;
08530           GETPTR;
08540           PTR^.NEXTPTR:=GRIDCHAR[P1,P2].GROUNDPTR;
08550           GRIDCHAR[P1,P2].GROUNDPTR:=PTR;
08560           PTR^.VALUE1:=TEMPGROUND;
08570           PTR^.VALUE2:=1;
08580           while PTR^.NEXTPTR#nil do PTR:=PTR^.NEXTPTR;
08590           TEMPPTR:=PTR;GETPTR;
08600           TEMPPTR^.NEXTPTR:=PTR
08610             end;
08620           %*************************************************************\
08630         procedure FILLADVGRID;
08640         (* puts the attribute of type of ground at the next grid point with direction *)
08650         %*************************************************************\
08660             begin
08670           if GRIDCHAR[CURRENTPOInter[1],CURRENTPOInter[2]].TYPOFGROUND=8 then
08680           FILLGRIDChar(CURRENTPOInter[1],CURRENTPOInter[2])
08690           else COMBINETYPofground(CURRENTPOinter[1],CURRENTPOinter[2]);
08700           PTR^.VALUE1:=(DIRECTION+4)mod(8)
08710             end;
08720           %*************************************************************\
08730         procedure FILLPRESENTgrid;
08740         (* puts the attribute of type of ground at the present point with direction *)
08750         %*************************************************************\
08760             begin
08770           if PTRSPOINT then
08780           if GRIDCHAR[LASTPOINTEr[1],LASTPOINTEr[2]].TYPOFGROUND=8 then
08790           COMBINETYPofground(LASTPOINTEr[1],LASTPOINTEr[2])
08800           else FILLGRIDChar(LASTPOINTEr[1],LASTPOINTEr[2])
08810           else
08820             begin
08830           if GRIDCHAR[LASTPOINTEr[1],LASTPOINTEr[2]].TYPOFGROUND=1 then
08840           PTR:=GRIDCHAR[LASTPOINTEr[1],LASTPOINTEr[2]].GROUNDPTR
08850           else
08860             begin
08870               PTR:=GRIDCHAR[LASTPOINTEr[1],LASTPOINTEr[2]].GROUNDPTR;
08880               while PTR^.NEXTPTR#nil do PTR:=PTR^.NEXTPTR
08890             end;
08900             end;
08910           PTR^.VALUE2:=DIRECTION;
08920           if I<3 then
08930             begin
08940           TEMPPTR:=PTR;GETPTR;
08950           PTR^.VALUE1:=WIDTH;
08960           PTR^.VALUE2:=DEPTH;TEMPPTR^.NEXTPTR:=PTR
```

```
08970             end
08980             end;
08990         %*************************************************************\
09000       procedure READCHAR;(*READS A CHARACTER FROM THE FILE MAPDAT *)
09010       %*************************************************************\
09020             var
09030           ALP:char;
09040             begin
09050           repeat READ(MAPDAT,CH)
09060           until CH#' ';
09070           while not (EOLN(MAPDAT)or(ALP='.')) do READ(MAPDAT,ALP);
09080             end;
09090         %*************************************************************\
09100       procedure CONVERTDEGree(var DIRECTION:RANGE);
09110       (* converts DEGREE to direction and outputs it in DIRECTION *)
09120       %*************************************************************\
09130             begin
09140           case DEGREE of
09150           0:DIRECTION:=0;45:DIRECTION:=1;
09160           90:DIRECTION:=2;135:DIRECTION:=3;
09170           180:DIRECTION:=4;225:DIRECTION:=5;
09180           270:DIRECTION:=6;315:DIRECTION:=7
09190           end;
09200             end;
09210         %*************************************************************\
09220       procedure READINCHAR;(*READS DIRECTLY INPUT UPTO 12 CHARACTERS FROM FILE MAPDAT*)
09230       %*************************************************************\
09240             var
09250           INCH:char;I:integer;
09260             begin
09270           for I:=1 to 12 do WORD[I]:=' ';
09280           INCH:=' ';I:=1;
09290           while INCH=' ' do READ(MAPDAT,INCH);
09300           while ((not EOLN(MAPDAT))and(I<13)and(INCH#'.')) do
09310             begin
09320           WORD[I]:=INCH;I:=I+1;READ(MAPDAT,INCH);
09330             end;
09340           if I>12 then MSG(3)
09350           else
09360           if EOLN(MAPDAT)then WORD[I]:=INCH
09370             end;
09380         %*************************************************************\
09390       procedure READNUMBER;
09400       %*************************************************************\
09410             begin
09420           READINCHAR;
09430           NUMBER
09440             end;
09450         %*************************************************************\
09460       procedure SPLIT(var POINT:SPLITBREAK);
09470       %*************************************************************\
09480             begin
09490           POINT[1]:=INTVAL div 100;
09500           POINT[2]:=INTVAL mod 100
09510             end;
09520         %*************************************************************\
```

```
09530       procedure FILLGRIDHEIGHTS;
09540       (* completes the grid heights after the contours have been described *)
09550         type
09560       GRIDRECORD=record
09570             HT,NO:integer
09580           end;
09590         var
09600       STACK:array[0..100] of GRIDRECORD;
09610       TOP,I,J:integer;
09620       {*****************************************************************\
09630         procedure FILLGRID(FROMNO,TONO,HT:integer);
09640         (*fills the heights HT between pts FROMNO to TONO *)
09650         {*****************************************************************\
09660         var
09670       K:integer;
09680         begin
09690       if TOP=0 then MSG(2)
09700       else
09710         begin
09720             for K:=FROMNO to TONO do
09730             if GRIDCHAR[K,I].ALTITUDE=0 then
09740             GRIDCHAR[K,I].ALTITUDE:=HT;
09750             STACK[TOP].HT:=0;STACK[TOP].NO:=0;TOP:=TOP-1
09760         end;
09770         end;
09780       {*****************************************************************\
09790         procedure PUSH(HEIGHT,NUMBER:integer);
09800         (*stores the height HEIGHT in stack for matching *)
09810         {*****************************************************************\
09820         var
09830       PUSHSTACK:boolean;
09840         begin
09850       PUSHSTACK:=false;
09860       if TOP<2 then PUSHSTACK:=true
09870       else
09880         begin
09890             if ((STACK[TOP].HT>STACK[TOP-1].HT)and(HEIGHT>STACK[TOP].HT))
09900             then PUSHSTACK:=true
09910             else
09920             if ((STACK[TOP].HT<STACK[TOP-1].HT)and(HEIGHT<STACK[TOP].HT))
09930             then  PUSHSTACK:=
09940             true
09950             else
09960             if HEIGHT=STACK[TOP-1].HT then
09970         begin FILLGRID(STACK[TOP-1].NO,NUMBER,HEIGHT);STACK[TOP].HT:=0;
09980             STACK[TOP]
09990             .NO:=0;TOP:=TOP-1;
10000         end
10010             else MSG(1);
10020         end;
10030       if PUSHSTACK then
10040         begin
10050             TOP:=TOP+1;
10060             if TOP >10 then MSG(1)
10070             else
10080         begin
```

```
10090                 with STACK[TOP] do
10100                   begin
10110                   HT:=HEIGHT;VD:=VIEWER;
10120                   end;
10130             end;
10140             end;
10150             end;
10160           {*****************************************************************}
10170             procedure INITSTACK;
10180             (* initialises the stack for starting *)
10190           {*****************************************************************}
10200             var
10210           I:integer;
10220             begin
10230           for I:=0 to 10 do with STACK[I] do
10240             begin
10250                 HT:=0;VD:=0
10260             end;
10270           TOP:=0;
10280             end;
10290             begin
10300           for I:=1 to 40 do
10310             begin
10320           INITSTACK;
10330           for J:=1 to 40 do
10340             begin
10350                 if GRIDCHAR[J,I].ALTITUDE>0 then
10360             begin
10370                 if STACK[TOP].HT>GRIDCHAR[J,I].ALTITUDE then
10380                 PUSH(GRIDCHAR[J,I].ALTITUDE,J)
10390                 else FILLGRID(STACK[TOP].VD,J,I,STACK[TOP].HT)
10400             end;
10410             end;
10420             end;
10430             end;
10440         begin
10450           for I:=1 to 7 do
10460             begin
10470           case I of
10480           1:FIXEDWORD:='RIVER      ';
10490           2:FIXEDWORD:='CANAL      ';
10500           3:FIXEDWORD:='ROAD       ';
10510           4:FIXEDWORD:='CROP/FIELD ';
10520           5:FIXEDWORD:='MINE FIELD ';
10530           6:FIXEDWORD:='MARSHY GO  ';
10540           7:FIXEDWORD:='CONTOUR    ';
10550           end;
10560           loop
10570           READCHAR;ERROR:=false;
10580           exit if CH='Y';
10590           if I<3 then
10600             begin
10610               READ(MAPDAT,WIDTH);
10620               READ(MAPDAT,DEPTH);
10630             end
10640           else
```

```
10650           if I=7 then
10660             begin
10670               READ(MAPDAT,HEIGHT)
10680             end;
10690           READNUMBER;FIRSTPOINT:=true;
10700           SPLIT(LASTPOINTer);
10710           loop
10720               READNUMBER;
10730           exit if ((ZERO='FINISH      ')or(not ISNUMBER)or(ERROR));
10740                 SPLIT(NEXTPOINTer);
10750                 GETDIRECTION(LASTPOINTEr[1],NEXTPOINTer[1],LASTPOINTEr[2],
10760                 NEXTPOINTer[2],DIRECTION);
10770                 repeat
10780                 CURRENTPOInter:=LASTPOINTEr;
10790                 CUR1:=CURRENTPOInter[1];CUR2:=CURRENTPOInter[2];
10800                 CALCNEXTGRID(CUR1,CUR2,DIRECTION);
10810                 CURRENTPOInter[1]:=CUR1;CURRENTPOInter[2]:=CUR2;
10820                 if ((CURRENTPOInter[1]<41)and(CURRENTPOInter[1]>0)
10830                 and(CURRENTPOInter[ 2]<41)and(CURRENTPOInter[2]>0)) then
10840                   begin
10850                 if I<7 then
10860                   begin
10870                 FILLADVGRID;FILLPRESENTGRID;
10880                   end
10890                 else GRIDCHAR[LASTPOINTEr[1],LASTPOINTEr[2]],ALTITUDE
10900                 :=HEIGHT;
10910                 LASTPOINTEr:=CURRENTPOInter;FIRSTPOINT:=false;
10920                   end
10930                 else
10940                   begin
10950                 MSG(11);
10960                 ERROR:=true;
10970                   end
10980                 until
10990                 ((CURRENTPOInter=NEXTPOINTEr)or (ERROR));
11000                 if CURRENTPOInter<>NEXTPOINTEr then MSG(12);
11010           end;
11020           end;
11030             end;
11040           FILLALTITUde;
11050           WRITELN(CITY,'IMPORTANT:note that the map details have been taken
11060     from your file = "MAPDAT" =');
11070           SKIP(2);
11080       end;
11090         {*****************************************************************\
11100       function ISVISIBLE(LOC1X,LOC1Y,LOC2X,LOC2Y:integer;CH:char):boolean;
11110       (* checks if the loc1 and loc2 are visible taking into consideration all other fact
11120        except that of line of sight *)
11130       {*****************************************************************\
11140       label
11150           10;
11160       var
11170           FINISH,RIGHT:boolean;
11180           DISTANCE,FINE:integer;
11190           {*****************************************************************\
11200       function CHECKVISIBility(LOC1X,LOC1Y,LOC2X,LOC2Y:integer):boolean;
```

```
11210       (* checks if the line of sight visibility is there or not *)
11220       %*************************************************************\
11230         var
11240       CHECKX,LOC1:boolean;
11250       INCR,STARTX,STARTY,ENDX,ENDY,CURRENTY,CURRENTX:integer;
11260       CURRANGLE,ANGLE,C,M:real;
11270       %*************************************************************\
11280         procedure CALNEXTX;
11290         %*************************************************************\
11300         begin
11310       CURRENTX:=CURRENTX + INCR;
11320       CURRENTY:=TRUNC(M*CURRENTX+C+0.5)
11330         end;
11340       %*************************************************************\
11350         procedure CALNEXTY;
11360         %*************************************************************\
11370         begin
11380       CURRENTY:=CURRENTY + INCR;
11390       CURRENTX:=TRUNC(((CURRENTY-C)/M)+0.5)
11400         end;
11410         begin
11420       if ABS(LOC1X-LOC2X) > ABS(LOC1Y-LOC2Y) then CHECKX:=true
11430       else CHECKX:=false;
11440       M:=(LOC2Y-LOC1Y)/(LOC2X-LOC1X);
11450       C:=LOC1Y-M*LOC1X;
11460       if GRIDCHAR[LOC1X,LOC1Y].ALTITUDE > GRIDCHAR[LOC2X,LOC2Y].ALTITUDE then
11470       LOC1:=true
11480       else
11490       LOC1:=false;
11500       CALCULATEDist(LOC1X,LOC1Y,LOC2X,LOC2Y,DISTANCE);
11510       if LOC1 = true then
11520         begin
11530       STARTX:=LOC2X;STARTY:=LOC2Y;
11540       ENDX:=LOC1X;ENDY:=LOC1Y;
11550         end
11560       else
11570         begin
11580       STARTX:=LOC1X;STARTY:=LOC1Y;
11590       ENDX:=LOC2X;ENDY:=LOC2Y;
11600         end;
11610       CURRENTX:=STARTX;CURRENTY:=STARTY;
11620       ANGLE:=(GRIDCHAR[STARTX,STARTY].ALTITUDE - GRIDCHAR[ENDX,ENDY].ALTITUDE)/
11630       DISTANCE;
11640       if CHECKX then
11650       if ENDX > STARTX then INCR:=1
11660       else INCR:=-1
11670       else
11680       if ENDY > STARTY then INCR:=1
11690       else INCR:=-1;
11700       CURRANGLE:=0.0;
11710       while (CURRANGLE<ANGLE)and((CURRENTX#ENDX)or(CURRENTY#ENDY))do
11720         begin
11730       if CHECKX then CALNEXTX
11740       else CALNEXTY;
11750       CALCULATEDist(CURRENTX,CURRENTY,ENDX,ENDY,DISTANCE);
11760       CURRANGLE:=(GRIDCHAR[CURRENTX,CURRENTY].ALTITUDE - GRIDCHAR[ENDX,ENDY].
```

```
12330             end
12340           else
12350             begin
12360                 ERROR(1);
12370                 ERR:=true;
12380             end
12390             end
12400             end
12410           else
12420             begin
12430           READWORD;GETREF;
12440           if ISNUMREF then
12450             begin
12460           LOC2X:=TNKDAT[TANKNO,NUMWORD[1],NUMWORD[2]].MAPCORD.X;
12470           LOC2Y:=TNKDAT[TANKNO,NUMWORD[1],NUMWORD[2]].MAPCORD.Y
12480             end
12490           else ERR:=true
12500             end
12510           ;
12520           if not ERR then
12530             begin
12540           case CH of
12550           'L':WRITELN(TTY,'LANDMARK',TEMP,'LOCATION',LOC2X,LOC2Y);
12560           'T':WRITELN(TTY,'TANK','LOCATION',LOC2X,LOC2Y);
12570           others:
12580           end;
12590           if ISVISIBLE(LOC1X,LOC1Y,LOC2X,LOC2Y,CH) then WRITELN(TTY,'IS VISIBLE FOR ME')
12600           else WRITELN(TTY,'IS NOT VISIBLE TO ME');
12610             end;
12620       end;
12630         {*****************************************************************\
12640       procedure REPORT( CH:char);(*REPORTS ACTIVITY OF ENEMY OR LOCATION OR FUEL BALANCE*)
12650       (* output details like FUEL BALANCE,LOCATION taking the
12660     details from the record TNKDAT for the tank IK,JK *)
12670       {*****************************************************************\
12680       begin
12690           WRITELN(TTY,'TANK ',JK:2,' OF TROOP ',IK:2,' REPORTING':12);WRITELN(TTY);
12700           if CH='F' then
12710             begin WRITE(TTY,'FUEL BALANCE:':25);WRITELN(TTY,TNKDAT[IK,JK].FUELCAP:6,'  LI
12720           )
12730             end;
12740           if CH='L' then
12750             begin WRITE(TTY,'LOCATION':10);WRITELN(TTY,'X':10,'Y':10);
12760           CONVERTGRIDref(GRX,GRY,TNKDAT[IK,JK].MAPCORD.X,TNKDAT[IK,JK].MAPCORD.Y);
12770           WRITELN(TTY,GRX:21,GRY:10)
12780             end;
12790           if CH='E' then
12800             begin WRITELN(TTY,'ENEMY ACTIVITY');WRITELN(TTY,'ENEMY SEEN AT LOC 628105')
12810             end
12820       end;
12830         {*****************************************************************\
12840       procedure   REPORTAMMunation:(*REPORTS BALANCE OF AMMUNITION OF ALL TYPES*)
12850       (* selects the type of ammunation balance for the tank IK,JK and  print the balance
12860       {*****************************************************************\
12870       var
12880           AP,SM,H,ALL:boolean;I,J:integer
```

```
12890         ;
12900       begin
12910         READWORD;
12920         if WORD='                ' then
12930           begin
12940         AP:=false;H:=false;SM:=false;
12950         repeat
12960         if WORD='AP              ' then
12970           begin AP:=true;READWORD
12980           end;
12990         if WORD='HE              ' then
13000           begin H:=true;READWORD
13010           end;
13020         if WORD='SMOKE           ' then
13030           begin  SM:=true;READWORD
13040           end;
13050         if WORD='ALL             ' then
13060           begin ALL:=true;READWORD
13070           end
13080         until (WORD<>'AP              ')or(WORD<>'SMOKE           ')or(WORD<>'HE              ');
13090           end
13100         else  ALL:=true;
13110         WRITELN(TTY,'TANK':10,IK:6,'OF TROOP':10,JK:6,'REPORTING AMMUNATION BALANCE':32)
13120         WRITELN(TTY,'TYPE':20,'QUANTITY':20);
13130         if ((AP=true)or(ALL=true)) then WRITELN(TTY,'AP':19,TNKDAT[IK,JK].AMNCAP.TYP1:2
13140         if (SM=true)or(ALL=true)
13150         then
13160         WRITELN(TTY,'SMOKE ':20,TNKDAT[IK,JK].AMNCAP.TYP2:19);
13170         if (H=true)or(ALL=true) then WRITELN(TTY,'HE  ':20,TNKDAT[IK,JK].AMNCAP.TYP3:19)
13180       end;
13190         {*****************************************************\
13200       procedure TANKSTATUS( TA:boolean);(*REPORTS STATUS OF OWN TANK OR OTHER TANK*)
13210       (* depending on the type of command reports the STATUS of self or of other tanks
13220        like BOGGED,MOVING etc *)
13230       {*****************************************************\
13240       var
13250         J,K,I:integer;TEMP:packed array[1..4]of char;LOCREC:boolean;
13260       begin LOCREC:=false;
13270         if TA=true then
13280           begin   READWORD;
13290         if WORD='LOC             ' then
13300           begin
13310         READWORD; for K:=1 to 4 do TEMP[K]:=WORD[K];
13320         LOCREC:=true
13330           end
13340         else
13350           begin
13360         NUMBER;J:=NUMWORD[2];I:=NUMWORD[1];READWORD;
13370         if WORD='LOC             ' then
13380           begin READWORD;for K:=1 to 6 do TEMP[K]:=WORD[K];
13390             LOCREC:=true
13400           end
13410           end
13420           end;
13430         WRITELN(TTY,'TANK':10,IK:3,'OF TROOP':10,JK:3);
13440         if TA=true then
```

```
13450             begin IK:=I; JK:=J;
13460           WRITELN(TTY,'REPORTING STATUS OF TANK':30,I:3,'OF ERROR':10,J:3);
13470           if LOCREC=true then WRITELN(TTY,'located at grid ref  ',TEMP)
13480             end
13490           else
13500           WRITELN(TTY,'REPORTING STATUS OF SELF':30);
13510           with TANKARR[IK,JK] do with STATUS do
13520             begin
13530           SKIP(2);
13540           if MUGGED then MSG(4);
13550           if MOVING then MSG(5)
13560           else WRITELN(TTY,'STANDING AT LOC ',MAPCORD.X,MAPCORD.Y);
13570           if FLAG then MSG(6);
13580           if not CANSHOOT then MSG(7);
13590           if not CANMOV then
13600             begin
13610                 MSG(8);
13620                 if SHOCKED then MSG(9)
13630                 else MSG(10)
13640             end
13650             end;
13660           SKIP(2)
13670       end;
13680           {*****************************************************************\
13690       procedure   REPORTPROC;(*ANALYISES THE REPORT PROCEDURE*)
13700       (* analyises the REPORT procedure and decides the type of report that is required *
13710       {*****************************************************************\
13720       begin READWORD;
13730           if WORD='STATUS       ' then
13740             begin READWORD;
13750           if WORD='.            ' then TANKSTATUS(false)
13760           else
13770           if WORD='IK           ' then TANKSTATUS(true)
13780           else ERROR(9)
13790             end
13800           else
13810             begin
13820           if WORD='LOCATION     ' then REPORT('L')
13830           else
13840           if WORD='BALANCE      ' then
13850             begin READWORD;
13860           if WORD='AMMUNATION   ' then REPORTAMMUnation
13870           else
13880           if WORD='FUEL         ' then REPORT('F')
13890           else ERROR(8)
13900             end
13910           else
13920           if WORD='ENEMY        ' then REPORT('E')
13930           else
13940           if WORD='VISIBILITY   ' then
13950             begin READWORD;
13960           if WORD='UM           ' then REPORTVISIbility('U')
13970           else
13980           if WORD='IK           ' then REPORTVISIbility('I')
13990           else ERROR(10)
14000             end
```

```
14010            else ERROR(3)
14020              end
14030        end;
14040            {*************************************************************\
14050          procedure CALCULATEPos(A,B:integer);
14060        (*CALCULATES NEW POSITION WHEN THE TANK IS ON THE MOVE*)
14070          (* this is used  to calculate destination for MOVE
14080        when the move command specifies only DIRECTION and DISTANCE *)
14090          (* A and B are the present position. DEG gives the direction
14100        of move in degrees and DIST gives   the  distance to be   moved *)
14110          (* the destination coordinates are updated  through MQ^.DEST *)
14120            {*************************************************************\
14130          begin
14140              ANGLE:=(DEG/180)*3.14;
14150              MQ^.DEST.X:=A+TRUNC(DIST*SIN(ANGLE)/500);
14160              MQ^.DEST.Y:=B+TRUNC(DIST*COS(ANGLE)/500);
14170          end;
14180            {*************************************************************\
14190          procedure CHECKDIRECTion(var DEG:integer);(*CONVERTS CARDINAL DIRECTIONS TO DEGREES
14200          {*************************************************************\
14210          begin
14220              DEG:=400;
14230              if(WORD='NORTH      ')or(WORD='N          '))then DEG:=0;
14240              if(WORD='NORTH-EAST ')or(WORD='NE         '))then DEG:=45;
14250              if(WORD='EAST       ')or(WORD='E          ')) then DEG:=90;
14260              if(WORD='SOUTH-EAST ')or(WORD='SE         ')) then DEG:=135;
14270              if(WORD='SOUTH      ')or(WORD='S          ')) then DEG:=180;
14280              if(WORD='SOUTH-WEST ')or(WORD='SW         ')) then DEG:=225;
14290              if(WORD='WEST       ')or(WORD='W          ')) then DEG:=270;
14300              if(WORD='NORTH-WEST ')or(WORD='NW         ')) then DEG:=315
14310          end;
14320            {*************************************************************\
14330          procedure   FILLREST(MQ:tab;(*COMPLETES THE MOVE TABLE*)
14340          (* copies the rest of the data  of tank from TNKDAT after
14350        checking its validity   of MOVE command. *)
14360          (* the return of ROGER ! segnifies that this tank is added to the MOVE list *)
14370          {*************************************************************\
14380          begin
14390              MQ^.SOURCE.X:=TNKDAT[IK,JK].MAPCORD.X;
14400              MQ^.SOURCE.Y:=TNKDAT[IK,JK].MAPCORD.Y;
14410              MQ^.DOC.X:=TNKDAT[IK,JK].MAPCORD.X;
14420              MQ^.DOC.Y:=TNKDAT[IK,JK].MAPCORD.Y;
14430              MQ^.STATE:=true;MLAST^.GLINK:=MQ;
14440              TNKDAT[IK,JK].STATUS.MOVING:=true;
14450              MQ^.CURROBJECTive:=1;
14460              MQ^.GLINK:=NLAST;MLAST:=MQ;WRITELN(TTY,'ROGER !');SKIP(1)
14470          end;
14480            {*************************************************************\
14490          procedure CHECKLANDMark;(*CHECKS IF THE LANDMARK EXISTS IN THE TABLE OF LANDMARKS*)
14500          (* checks LANMARKS so that the same name landmarks are not redeclared.
14510            (* it checks if the landmark exist and returns its addref in G4 and sets
14520            ISLANDMARK as true   *)
14530          {*************************************************************\
14540          begin
14550              ISLANDMARK:=false;G4:=0;
14560              if LMROOT^.RLINK<>nil then
```

```
14570             begin
14580           LP:=LMP.FIRST;
14590           while (LP^.NEXT<>NIL)and(LP<>nil) do
14600             begin
14610           if WORD>LP^.NAME then LP:=LP^.RLINK
14620           else LP:=LP^.LLINK
14630             end;
14640           if LP^.NAME=WORD then
14650             begin
14660           ISLANDMARK:=true;GR:=LP^.GR
14670             end;
14680
14690             end
14700       end;
14710         {*****************************************************************\
14720       procedure CHECKGRIDRef(var A,B:integer);
14730       {* checks the tokens ,firstly if the WORD is a NUMBER or LANDMARK,if it is
14740         then converts and gets its griderf in A, B *)
14750       {*****************************************************************\
14760       var
14770           G:integer;
14780       begin
14790           G:=GR;
14800           if not ISNUMBER then CHECKLANDMark(G);
14810           if ISNUMBER or ISLANDMARK then
14820             begin
14830           if ISNUMBER then CONVERT(INTVAL,A,B)
14840           else CONVERT(G,A,B)
14850             end
14860       end;
14870         {*****************************************************************\
14880       procedure ENTERROUTE:(*ENTERS THE ROUTE TO BE TAKEN BY THE BANK ON THE MOVE*)
14890       (* this is part of the MOVE table filling. it fills the route to be taken
14900         by the tank if specified in input command. *)
14910       (* max points are limited to 10. the no of points are kept in NO^.OBJECTIVE *)
14920       (* the current point is used at the time of processing and
14930     is stored in NO^.CURROBJECTIVE *)
14940         {*****************************************************************\
14950       var
14960           J,I,G:integer;A,B:integer;
14970       begin
14980           I:=0;for J:=1 to 10 do
14990             begin
15000           NO^.ROUTE[J].X:=0;
15010           NO^.ROUTE[I].Y:=0
15020             end;
15030           loop
15040           READWORD;
15050           exit if ((WORD='.')or(I=11));
15060           CHECKGRIDRef(A,B);
15070           if (ISNUMBER or ISLANDMARK)then
15080             begin
15090           I:=I+1;NO^.ROUTE[I].X:=A;NO^.ROUTE[I].Y:=B
15100             end
15110           else ERROR(18)
15120           end;
```

```
15130          [illegible]
15140          if not ISI[illegible]AB then FILLRESTMOvtab
15150        end;
15160        {*****************************************************************\
15170        procedure CALCROUTEonvtab;(*ANALYISES THE DISTANCE OF MOVE COMMAND*)
15180        (* this processes 2nd branch of the move command where DIRECTION and DISTANCE
15190         is used to describe new location *)
15200        {*****************************************************************\
15210        var
15220            C ,A,B:integer;
15230        begin
15240            NUMBER;DEG:=VAL;
15250            if ISNUMBER then
15260              begin
15270            DIST:=INTVAL;READWORD;
15280            if ((WORD='METERS          ')or(WORD='MTRS          ')or(WORD='MTR           ')
15290              or(WORD='KM              ')or( WORD='KMS           '))
15300            then
15310              begin
15320            if ( (WORD='KM              ') or ( WORD='KMS           '))
15330            then DIST:=DIST*1000;
15340            READWORD;(*METER*);
15350            if((ISNUMBER)and((INTVAL<361)) then
15360              begin
15370                DEG:=INTVAL;READWORD
15380              end
15390            else
15400            if not ISNUMBER then CHECKDIRECtion(DEG)
15410            else ERROR(13);
15420            if DEG<361 then
15430              begin
15440                READWORD;
15450                if WORD='.               ' then
15460              begin
15470                CALCVALUEPosn(TNKDATA[IK,JK].MAPCORD.X,
15480                TNKDATA[IK,JK].MAPCORD.Y);
15490                FILLRESTMOvtab
15500              end
15510                else
15520                if WORD='OF              ' then
15530              begin READWORD; CHECKGRIDRef(A,B);
15540                if  ISNUMBER or ISLANDMARK then
15550                  begin
15560
15570                CALCVALUEPosn(A,B);READWORD;
15580                if
15590
15600                WORD='ROUTE         ' then ERRROUTE
15610                else FILLRESTMOvtab
15620                  end
15630                else ERROR(14)
15640              end
15650                else
15660                ERROR(15)
15670              end
15680            else ERROR(15)
```

```
            end
          else ERROR(17)
            end;
      end;
          %*****************************************************************\
      procedure ENTERMOVTAB;(*ANALYISES THE OBJECTIVE PART OF THE MOVE COMMAND*)
      (* this processes the branch of MOVE where a straight
    forward move to objective is there *)
      %***************************************************************\
      var
          A:integer;B:integer;
      begin
          READWORD;CHECKREFORef(A,B);
          if ISCOMPANY or ISLANDMARK then
            begin
          MR^.DEST.X:=A;MR^.DEST.Y:=B;READWORD;
          if WORD='.                    ' then FILLRESTMOVTAB
          else
          if WORD='ROUTE                ' then ENTERROUTE
          else ERROR(20)
            end
          else ERROR(21)
      end;
          %*****************************************************************\
      procedure DELETEMOVTAB(TEMPPTR:MPTR);(*REMOVES ATANK FROM THE MOVE TABLE*)
      (* this deletes an entry in movetable addressed by TEMPPTR . used at reaching
       of destination or nuetralisation or re-entering of table *)
      %***************************************************************\
      var
          TEMP:MPTR;
      begin
          TEMP:=TEMPPTR^.ULINK;TEMP^.LLINK:=TEMPPTR^.LLINK;
          TEMP:=TEMPPTR^.LLINK;
          if TEMP<>nil then TEMP^.ULINK:=TEMPPTR^.ULINK
          else MLAST:=TEMPPTR^.ULINK
      end;
          %*****************************************************************\
      procedure CHECKMOVTAB;(*CHECKS IF THE TANK IS ALREADY ON THE MOVE*)
      %***************************************************************\
      begin
          ISINMOVTAB:=false;
          MR:=MOVTABROOT;while ((MR^.TANKNO#TANKNUMBER)and(MR<>MLAST)) do
          MR:=MR^.LLINK;
          if MR^.TANKNO=TANKNUMBER then ISINMOVTAB:=true
      end;
          %*****************************************************************\
      procedure MOVEPROCEDure;
      %***************************************************************\
          forward;
          %*****************************************************************\
      procedure CHANGEDESTination;(*CHANGE DESTINATION WHEN THE TANK IS ON THE MOVE*)
      %***************************************************************\
      var
          A,B:integer;
      begin
          WRITE(TTY,'NEW DESTINATION:');BREAK;
```

```
16250           READINPUT;
16260           CHECKGRIDREF(X,Y);
16270           if ISNUMBER or ISALPHAMARK then
16280             begin
16290             ^.DEST.X:=X;^.DEST.Y:=Y;^.STATE:=true;
16300             ^.DEST.X:=X;
16310               end
16320             else
16330               begin
16340               ERROR(21);
16350               CHANGEDESTINATION
16360               end
16370         end;
16380       {*****************************************************************}
16390       procedure CHANGEROUTE;(*CHANGES ROUTE OF A TANK ON MOVE*)
16400       {*****************************************************************}
16410       begin
16420           WRITELN(TTY,'NEW ROUTE:');BREAK;
16430           READLINE;X:=R;
16440           X^.STATE:=true;
16450           ENTERROUTE
16460       end;
16470       {*****************************************************************}
16480       procedure MOVTABCHANGE;(*CHANES THE MOVE TABLE*)
16490       {*****************************************************************}
16500       begin
16510           DELETEMOVTAB(OR);
16520           WRITE(TTY,'CHANGE TO-MOVE:');BREAK;READLINE;
16530           MOVEPROCEDURE
16540       end;
16550       {*****************************************************************}
16560       procedure ALTERMOVTAB;(*CHECKS THE PART TO BE CHANGED IN MOVE TABLE*)
16570       {*****************************************************************}
16580       begin
16590           WRITE(TTY,'TANK ALREADY ON MOVE. DO YOU WANT TO CHANGE STATUS ?');BREAK;
16600           READCH;
16610           if CH='Y' then
16620             begin
16630             WRITELN(TTY,'FOR CHANGING ONLY DESTINATION TYPE "D"');
16640             WRITELN(TTY,'FOR CHANGING ONLY ROUTE TYPE "R"');
16650             WRITELN(TTY,'FOR COMPLETE CHANGING "C"');
16660             WRITELN(TTY,'FOR RETURN WITHOUT CHANGE "N"');
16670             WRITE(TTY,'TYPE OF CHANGE REQUIRED ?');
16680             BREAK;READCH;
16690             case CH of
16700             'D':CHANGEDESTINATION;
16710             'R':CHANGEROUTE;
16720             'C':MOVTABCHANGE;
16730             'N':;
16740             others:MSG(2)
16750             end
16760               end
16770       end;
16780       {*****************************************************************}
16790       procedure MOVEPROCEDURE;(*RESUMES MOVE OF A TANK*)
16800       {*****************************************************************}
```

```
16810       begin
16820           CHECKMOVTAB;READKBD;
16830           if ISMOVTAB then
16840             begin
16850           if KBD=',                ' then
16860             begin
16870           if MR^.STATE then MSG(1);
16880           MR^.STATE:=true
16890             end
16900           else ALTERMOVTAB
16910             end
16920           else
16930             begin
16940           NEW(MR);MR^.ULINK:=nil;MQ^.TANKNO:=TANKNUMBER;MR^.OBJECTIVE:=0;
16950           if KBD='1                ' then ENTERMOVTABobj
16960           else ENTERDISTabtab
16970             end
16980       end;
16990           {*****************************************************************\
17000       procedure PRINTMOVTAB;(*PRINTS THE MOVE TABLE*)
17010       {*****************************************************************\
17020       begin
17030           WRITELN(LIY,'TANK NO':12,'DESTINATION':20,'STATE':15,'OBJECTIVE':13);
17040           MR:=MOVTABROOT;
17050           while MR<>NIL do
17060             begin
17070           MR:=MR^.ULINK;
17080           WRITELN(LIY,MR^.TANKNO:12,MR^.DEST.X:10,MR^.DEST.Y,MR^.STATE:12,
17090             MR^.OBJECTIVE:10)
17100             end;
17110       end;
17120           {*****************************************************************\
17130       procedure DELETEFTAB(FTEMP:FPTR);(*deletes a record with ptr FTEMP *)
17140       {*****************************************************************\
17150       begin
17160           if FTEMP=FLAST then
17170             begin  FLAST:=FTEMP^.ULINK;
17180           FTEMP^.ULINK^.LLINK:=nil;
17190             end
17200           else
17210             begin
17220           FTEMP^.ULINK^.LLINK:=FTEMP^.LLINK;
17230           FTEMP^.LLINK^.ULINK:=FTEMP^.ULINK;
17240             end;
17250       end;
17260           {*****************************************************************\
17270       function FIRETABPTR(I,J:integer):FPTR;
17280       (* checks if the tank I,J is already firing , if it is then
17290     returns its ptr value fPTR *)
17300       {*****************************************************************\
17310       var
17320           FTEMP:FPTR;
17330       begin FTEMP:=FROOT;FIRETABPTR:=nil;
17340           while FTEMP<>FLAST do
17350             begin FTEMP:=FTEMP^.LLINK;
17360           if ((FTEMP^.SOURCEIK[1]=I)and(FTEMP^.SOURCEIK[2]=J))then
```

```
17370             begin
17380           ARTREQ(TIV,'THIS TANK ALREADY FIRING,BALANCE OF',FIRES^.TOTAL,
17390             'ROUNDS WERE FIRED');
17400           ARTREQ(TIV,'DO YOU WANT TO CHANGE TARGET ?:');
17410           if YES then OBJECTNO(TEMP)
17420           else FIRES[A,B]:=FP;
17430             end
17440             end;
17450       end;
17460           {*****************************************************************\
17470       procedure FIREPROCEDure;
17480       (*FIRES the shells that are required to be fired be tanks and checks its effect *)
17490       {*****************************************************************\
17500       label
17510           15;
17520       var
17530           X,Y:integer;TEMP:POINTERWORD;
17540           {*****************************************************************\
17550       function PROBABILITY(I,J:integer):real;
17560       (* calculates the probability of hit depending on move etc of the tank *)
17570       {*****************************************************************\
17580             var
17590           VALUE:real;
17600             begin
17610           with INKDAT[I,J].STATUS do
17620             begin
17630           PROBABILITY:=0.98;VALUE:=1.0;
17640           if BOGGED then
17650             begin VALUE:=0.8;PROBABILITY:=0.3;
17660             end;
17670           if MOVING then
17680             begin VALUE:=0.98;PROBABILITY:=0.89;
17690             end;
17700           if FIRING then PROBABILITY:=0.97*VALUE;
17710           if CUPOLACLOSed then PROBABILITY:=0.9*VALUE;
17720             end
17730             end;
17740           {*****************************************************************\
17750       procedure JOINFIRE[4];
17760       (*COPIES the rest of the details and joins the record to the table after checking *)
17770       {*****************************************************************\
17780             begin
17790           FLAST^.GLINK:=FP;FP^.JLINK:=FLAST;
17800           FLAST:=FP;
17810           if FP^.DESTIK[1]#) then
17820           FP^.HITPROB:=PROBABILITY(IK,JK)*PROBABILITY(FP^.DESTIK[1],FP^.DESTIK[2])
17830           else FP^.HITPROB:=PROBABILITY(IK,JK);
17840           FP^.TIME:=INKDAT[IK,JK].GUNTIME;
17850             end;
17860           {*****************************************************************\
17870       procedure GETNEWNODe;
17880       {*****************************************************************\
17890             var
17900           I:integer;
17910             begin
17920           NEW(FP);
```

```
17930           FP^.SOURCETK[1]:=IK;
17940           FP^.SOURCETK[2]:=JK;
17950           FP^.SOURCELOC.X:=[illegible][IK,JK].MAPCORD.X;
17960           FP^.SOURCELOC.Y:=[illegible][IK,JK].MAPCORD.Y;
17970           if [illegible][IK,JK].STATUS.MOVING then FP^.RATE:=GUNCHAR[TYPOFTANK].[illegible]
17980           else FP^.RATE:=GUNCHAR[TYPOFTANK].RATEHALT;
17990           for I:=1 to 2 do FP^.DESTTK[I]:=0;
18000           FP^.[illegible]TK:=nil;
18010             end;
18020         {*****************************************************************\
18030       procedure GETLOCATION;
18040       (* gets the location of the tank in gridref *)
18050       {*****************************************************************\
18060             begin
18070         READWORD;
18080         TESTNUMBER:=true; NUMBER;
18090         if ISNUMBER then
18100           begin
18110         CONVERT(INTVAL,X,Y);
18120         FP^.DESTLOC.X:=X;FP^.DESTLOC.Y:=Y;
18130         JOIN:=true
18140           end
18150           end;
18160         {*****************************************************************\
18170       function CHECKOTHERdirection:boolean;
18180       {*****************************************************************\
18190             begin
18200         if ((WORD='BACK          ') or (WORD='FRONT         ') or (WORD='RIGHT         ')
18210             or (WORD='LEFT          ')
18220           )) then
18230           begin READWORD;
18240         if WORD='OF            ' then
18250           begin
18260               READWORD;
18270               if WORD='YOU           ' then
18280               CHECKOTHERdirection:=true;
18290           end;
18300           end
18310           end;
18320       begin
18330         if ((TNKDATA[IK,JK].STATUS.CANSHOOT)and(FIREIABPTR[IK,JK]=nil)) then
18340           begin
18350         JOIN:=false;
18360         TEMP:=WORD;TESTNUMBER:=true;
18370         READWORD;NUMBER;
18380         if ISNUMBER then
18390           begin
18400         GETNEWFNODe;FP^.TOFNUM:=INTVAL;
18410         READWORD;
18420         if WORD='SMOKE         ' then FP^.TYPOFAMM:=1
18430         else
18440         if WORD='HE            ' then FP^.TYPOFAMM:=2
18450         else
18460         if WORD='AP            ' then
18470           begin
18480               if TEMP='SHOOT         ' then MSG(1);
```

```
18490                 FP^.TYPE:=3
18500               end
18510             else
18520               begin ERROR(23);
18530                 goto 15
18540               end;
18550             READWORD;
18560             if (WORD='SHELLS       ') or (WORD='SHELLS       ') then
18570               begin
18580                 READWORD;
18590                 if WORD='AT           ' then
18600               begin
18610                 READWORD;
18620                 if ((WORD='TK          ')or(WORD='TANK        ')) then
18630                   begin
18640                 READNUMBER(NUMBER);
18650                 FP^.DESTTK[1]:=NUMWORD[1];
18660                 FP^.DESTTK[2]:=NUMWORD[2];
18670                 if ISNUMBER then
18680                   begin READWORD;
18690                 if WORD='ON           ' then GETLOCATION
18700                 else
18710                   begin
18720                     if ((not CHECKOTHERdirection)and(WORD#'.           ')) then
18730                     MSG(17);
18740                     FP^.DESTLOC.X:=TNKDAT[FP^.DESTTK[1],FP^.
18750                     DESTTK[2]].
18760                     MAPCORD.X;
18770                     FP^.DESTLOC.Y:=TNKDAT[FP^.DESTTK[1],FP^.
18780                     DESTTK[2]].
18790                     MAPCORD.Y;
18800                     JOIN:=true;
18810                   end;
18820                   end
18830                 else
18840                   begin
18850                 ERROR(24);
18860                 goto 15
18870                   end
18880                   end
18890                 else ERROR(25)
18900               end
18910                 else
18920                 if WORD='ON           ' then GETLOCATION
18930                 else ERROR(26);
18940               end
18950             else ERROR(27)
18960               end
18970             else MSG(18)
18980               end;
18990             if JOIN=true then JOINFIREIAb;
19000       15:
19010       end;
19020         {*****************************************************************\
19030       procedure PROCESSMOVtank(PTR:MPTR);
19040       (* processses the move of the tank , checking for the obstackles and taking action
```

```
19050       %*****************************************************************\
19060       const
19070           GRADIENTFactor=0.45;
19080       var
19090           DIRECTION:RANGE;RELCOUNT,CURRPIXEL,NEXTX,NEXTY:integer;FINISH:boolean;
19100           SPEED,DISTANCE:integer; FACTOR:real;FIRSTIME:boolean;
19110           %*****************************************************************\
19120       procedure CHANGEDIRECTion;
19130       (* gets the next point of objective once the grid pt has been reached *)
19140       %*****************************************************************\
19150           begin
19160           if PTR^.OBJECTIVE < PTR^.CURROBJECTive then
19170             begin
19180           NEXTX:=PTR^.ROUTEPTR^.X;
19190           NEXTY:=PTR^.ROUTEPTR^.Y
19200             end
19210           else
19220             begin
19230           NEXTX:=PTR^.ROUTEPTR^.CURROBJECTive1.X;
19240           NEXTY:=PTR^.ROUTEPTR^.CURROBJECTive1.Y
19250             end
19260             end;
19270           %*****************************************************************\
19280       procedure COMPUTETime;
19290       (* computes the time that the tank has taken to move from one pt to another *)
19300       %*****************************************************************\
19310           var
19320           X1,Y1,X2,Y2:integer;FACTOR:real;LPKM:1..5;
19330           TPTR:GRIDPTR;DISTANCE:integer;
19340           %*****************************************************\
19350             procedure CHECKCROSSobstacle(GROUND:RANGE);
19360             (* checks for the obstacle crossing for ground type GROUND *)
19370             %*****************************************************\
19380             begin LPKM:=5;
19390           if TANKCHAR[TYPOFTANK].HEIGHT < TPTR^.NEXTPTR^.VALUE2 then
19400             begin MSG(12);MSG(15);REACHEDESTination:=true;
19410                 with TANKDAT[IK,JK].STATUS do
19420             begin
19430                 BOGGED:=true;CANMOV:=false;CANSHOOT:=false
19440             end
19450             end
19460           else
19470             begin
19480                 if ((GROUND=2)and(TANKCHAR[TYPOFTANK].SPEEDUPTO <TPTR^.NEXTPTR^.VALUE2))
19490       then
19500           begin
19510                 MSG(12);MSG(15);FINISH:=true;
19520                 PTR^.STATE:=false
19530             end
19540                 else CROSSED:=true;
19550             end
19560             end;
19570           %*****************************************************\
19580             procedure REPORTWATERobstacle(GROUND:RANGE);
19590             (* reports the encountering of obstacle and asks for the action to be taken*)
19600             %*****************************************************\
```

```
19610             var
19620           WORD:packed array[1..3]of char;
19630           DIRWORD1,DIRWORD2:DIRECTION;
19640             begin
19650           if GROUND=1 then WORD:='RIVER'
19660           else WORD:='MARSH';
19670           DIRECTION(IPTR^.VALUE1,DIRWORD1);DIRECTION(IPTR^.VALUE2,
19680                     DIRWORD2);
19690           WRITELN(TTY,'TANK ',IK:2,JK:2,'  REPORTING FROM LOCATION ',X1:3,Y1:3,' '
19700             :2,WORD,'  OBSTACLE AHEAD');
19710           WRITELN(TTY,'WITH 'ALL' CHARACTERISTICS AS FOLLOWS:=');
19720           SKIP(1);
19730           WRITELN(TTY,'                                DIRECTION  ',DIRWORD1:12,'=>>    ',
19740             DIRWORD2:12);
19750           WRITELN(TTY,'APPROX.WIDTH  ',IPTR^.DEXPERTS^.VALUE1:4,'   METERS');
19760           WRITELN(TTY,'APPROX.DEPTH  ',IPTR^.DEXPERTS^.VALUE2:4,'   CMS');
19770           WRITELN(TTY,'SHOULD I ATTEMPT THE OBSTACLE or CHANGE COURSE ?');
19780           WRITE(TTY,'TYPE IN A "C" FOR CHANGING COURSE OR A "GO" TO CONTINUE:');
19790           BREAK;READCH;IR:=PTR;FLUSH:=true;
19800           if CH='C' then ALTERNATIVE
19810           else CHECKCROSSobstacle(GROUND)
19820             end;
19830           {*************************************************\
19840             procedure CHANGEFACTor;
19850             {*************************************************\
19860           forward;
19870           {*************************************************\
19880             procedure GROUNDFACTor(GROUND:RANGE);
19890             (* decides the type of action to be taken for ground GROUND *)
19900             {*************************************************\
19910             begin
19920           case GROUND of
19930                1,2:REPORTWATERobstacle(GROUND);
19940                3:
19950                if ((IPTR^.VALUE1 # DIRECTION) or
19960               (IPTR^.VALUE2 # DIRECTION)) then
19970                  begin
19980                LPKM:=2;FACTOR:=FACTOR*0.6;
19990                  end;
20000                4:FACTOR:=FACTOR*0.5;
20010                5:
20020                  begin
20030                MSG(12);
20040                WRITELN(TTY,'FROM LOCATION ',X2:2,Y2:2);
20050                WRITELN(TTY,'  TANK HAS RUN INTO ENEMY MINEFIELD!');
20060                REACHEDDESTination:=true;TNKDATTK,JK].STATUS.
20070                NEUTRALISED:=true
20080                  end;
20090                6:
20100                  begin
20110                MSG(12);
20120                WRITE(TTY,'  FROM LOCATION ',X2:2,Y2:2);
20130                WRITELN(TTY,'  TANK BOGGED DOWN IN MARSHY GROUND!');
20140                PTR^.STATE:=false;with TNKDAT[IK,JK],STATUS do
20150                  begin BOGGED:=true;CANMOV:=false;MOVING:=false
20160                  end;
```

```
20170                 REACHGOAL:=true
20180                   end;
20190                 7:GROUNDFACTor
20200           end
20210             end;
20220           %*****************************************\
20230             procedure GROUNDFACTor;
20240             %*****************************************\
20250             var
20260           I:integer;TEMP:;RIPTR;BRIDGE:boolean;
20270             begin
20280           TEMP:=TPTR;TPTR:=TPTR^.NEXTPTR;
20290           BRIDGE:=false;
20300           if ((TEMP^.VALUE2=TEMP^((TEMP^.VALUE1=1) or (TEMP^.VALUE1=2)))
20310           then BRIDGE:=true;
20320           if not BRIDGE then GROUNDFACTor(TEMP^.VALUE1);
20330           if not REACH then
20340             begin
20350                 if TEMP^.VALUE1 < 3 then for I:=1 to 2 do TPTR:=TPTR^.NEXTPTR
20360                 else TPTR:=TPTR^.NEXTPTR;
20370                 GROUNDFACTor(TEMP^.VALUE2)
20380             end
20390             end;
20400           %*****************************************\
20410             procedure CHECKSIDE;
20420             %*****************************************\
20430             var
20440           TEMPDIR:RANGE;TEMPTR:GRIDPTR;TEMPGROUND:RANGE;I:integer;
20450             begin
20460           TEMPGROUND:=GRIDCHAR[X2,Y2].TYPEGROUND;
20470           TEMPTR:=GRIDCHAR[X2,Y2].GROUNDPTR;
20480           TPTR:=TEMPTR;
20490           if TEMPGROUND # R then
20500             begin
20510                 if TEMPGROUND # 7 then
20520             begin
20530                 TEMPDIR:=(DIRECTION+2)mod(8);
20540                 if ((TEMPTR^.VALUE1=TEMPDIR)or(TEMPTR^.VALUE2=TEMPDIR))
20550                 then GROUNDFACTor(TEMPGROUND)
20560             end
20570                 else
20580             begin
20590                 TPTR:=TPTR^.NEXTPTR;
20600                 if ((TEMPTR^.NEXTPTR^.VALUE1=TEMPDIR)or
20610                (TEMPTR^.NEXTPTR^.VALUE2=TEMPDIR))
20620                 then GROUNDFACTor(TEMPTR^.VALUE1);
20630                 TEMPGROUND:=TEMPTR^.VALUE2;
20640                 if TEMPTR^.VALUE1 < 3 then
20650                 for I:=1 to 2 do TPTR:=TPTR^.NEXTPTR
20660                 else TPTR:=TPTR^.NEXTPTR;
20670                 if ((TEMPTR^.VALUE1=TEMPDIR)or(TEMPTR^.VALUE2=TEMPDIR))
20680                 then GROUNDFACTor(TEMPGROUND)
20690             end
20700             end
20710             end;
20720           %*****************************************\
```

```
20730            procedure GRADIENTFACTOR;
20740            (* calculates the gradient of move and its effect on speed *)
20750            %*************************************************\.
20760            var
20770          GRADIENT:integer;
20780            begin
20790          GRADIENT:=GRIDDATA[X2,Y2].ALTITUDE - GRIDDATA[X1,Y1].ALTITUDE;
20800          if GRADIENT < 0 then
20810          FACTOR:=FACTOR*(GRADIENT/50)*0.05
20820            end;
20830          %*************************************************\.
20840            procedure UPDATEFUEL;
20850            (* UPDATES THE FUEL BALANCE AFTER REDUCING ENCUMBERING AND THEN REDUCING IF *)
20860            (* updates the fuel that has been spent *)
20870          %*************************************************\.
20880            var
20890          BAL,FUEL:integer;
20900            begin
20910          FUEL:=TRUNC((DISTANCE*SPEED/1000)+0.5);
20920          BAL:=PLAYDATA[TK,JK].FUELCAP - FUEL;
20930          if BAL <= 0 then
20940            begin
20950                MSG(12);MSG(13);
20960                FINISH:=true;PTR^.SPAIR:=false
20970            end
20980          else
20990            begin
21000                PLAYDATA[TK,JK].FUELCAP:=BAL;
21010                if FIRSTIME then
21020                if BAL < PLAYCHAR[TYPOFPLAY].FIGURES then
21030            begin
21040                MSG(12);MSG(14);
21050                FIRSTIME:=false
21060            end
21070            end
21080            end;
21090            begin
21100          X1:=PTR^.LOC.X;Y1:=PTR^.LOC.Y;
21110          SPEED:=PLAYCHAR[I].SPEEDROAD;
21120          FACTOR:=1.0;
21130          DISTANCE:=500;
21140          case DIRECTION of
21150          1,3,5,7:
21160            begin
21170                X2:=X1;Y2:=Y1;
21180                CALLNEXTGRID(X2,Y2,DIRECTION-1);
21190                CHECKSIDE;DISTANCE:=700;
21200            end;
21210          others:
21220          end;
21230          if not FINISH then
21240            begin
21250          X2:=X1;Y2:=Y1;
21260          CALLNEXTGRID(X2,Y2,DIRECTION);
21270          TEMP:=GRIDDATA[X2,Y2].GRIDPOS;
```

```
21290           GROUNDFACTor(GRIDCHARLY2,X21,TYPEOFGROUND);
21300           GRADIENTFACtor;
21310             end;
21320           if ((not FINISH )or(CROSSED))then
21330             begin
21340           PTR^.LOC.X:=X2;PTR^.LOC.Y:=)2;
21350           SPEED:=TRUNC(SPEED* FACTOR);
21360           INKDATTIK,JKI.MAPCORD.X:=X2;INKDATTIK,JKI.MAPCORD.Y:=Y2;
21370           CURRENTIME:=TRUNC((36*DISTANCE)/(SPEED*10));
21380           UPDATEFUED;INKDATTIK,JKI.DRIVAS:=DIRECTI);
21390             end
21400             end;
21410       begin
21420           TIMECOUNT:=0;FINISH:=false;FIRSTTIME:=true;
21430           CHANGEDIRECtion;
21440           SKIP(3);CROSSED:=false;
21450           IK:=(PTR^.TANKID)div(10);JK:=(PTR^.TANKID)mod(10);
21460           CURRENTIME:=0;
21470           repeat
21480           GETDIRECTIon(PTR^.LOC.X,NEXTX,PTR^.LOC.Y,NEXTY,DIRECTION);
21490           COMPUTETIme;
21500           TIMECOUNT:=TIMECOUNT+CURRENTIME;
21510           if (PTR^.LOC.X=PTR^.DEST.X)and(PTR^.LOC.Y=PTR^.DEST.Y)then
21520             begin
21530           FINISH:=true;REACHEDESTination:=true;INKDATTIK,JKI.STATUS.MOVING:=false
21540             end;
21550           if not FINISH then
21560           if (NEXTX=PTR^.LOC.X)and(NEXTY=PTR^.LOC.Y) then
21570             begin
21580           PTR^.CURROBJECTive:=PTR^.CURROBJECTive+1;
21590           CHANGEDIRECtion
21600             end
21610           until ((TIMESTEP<(TIMECOUNT+0.5*CURRENTIME))or(FINISH)or(REACHEDESTination));
21620       end;
21630           {*****************************************************************\
21640       procedure MOVETANKS;
21650       (*moves the tanks from a location to another when command has been given *)
21660       {*****************************************************************\
21670       var
21680           TEMP,PTR:MPTR;
21690       begin
21700           PTR:=MOVTABROOT^.GLINK;
21710           while PTR<>nil do
21720             begin
21730           REACHEDESTination:=false;
21740           if PTR^.STATE then PROCESSMOVetank(PTR);
21750           TEMP:=PTR;
21760           PTR:=PTR^.GLINK;
21770           if REACHEDESTination then DELETEMOVTab(TEMP)
21780             end
21790       end;
21800           {*****************************************************************\
21810       procedure FIRE;
21820       (*FIRES the shell that has been commanded for the tank to fire *)
21830       {*****************************************************************\
21840       var
```

```
21850           J,PLUS,DISTANCE,NO,PENROTST,MAX,HIT:integer;
21860           PROB,ASINE,ACOS:real;
21870           LOCHIT:boolean;
21880           %*************************************************************\
21890       procedure CHECKHIT;
21900       (* CHECKS FOR THE EFFECT OF THE FIRING OF THE SHELL AND PASSES CONTROL ACCORDINGLY *)
21910       %*************************************************************\
21920           var
21930       II,JJ:integer;
21940       %*************************************************************\
21950           procedure ANALYSEHIT(I,J,HITTYPE,AMMTYPE:integer);
21960           (* analyises the hit in terms of its angle of hit,distance and where it hit*)
21970           %*************************************************************\
21980           var
21990       DISTYPE,ANGLETYPE,ANGLE:integer;
22000           function PENETRATED:boolean;
22010           var
22020               TYP,GUNSHOT,ARMR1,ARMR2,ARMR3:integer;
22030           begin
22040               ARMR1:=TANKCHAR[TNKDAT[I,JJ].TYPNO].ARMRFRONT;
22050               ARMR2:=TANKCHAR[TNKDAT[I,JJ].TYPNO].ARMRSIDE;
22060               ARMR3:=TANKCHAR[TNKDAT[I,JJ].TYPNO].ARMRREAR;
22070               TYP:=TNKDAT[I,JJ].TYPNO;
22080               PENETRATED:=false;
22090               if AMMTYPE=3 then
22100           begin
22110               case DISTYPE of
22120               0:GUNSHOT:=GUNCHAR[TYP].AR500;
22130               1:GUNSHOT:=GUNCHAR[TYP].AR1000;
22140               2:GUNSHOT:=GUNCHAR[TYP].AR2000;
22150               others:
22160               end;
22170               case ANGLE of
22180               1:
22190               if GUNSHOT> ARMR1 then PENETRATED:=true;
22200               2:
22210               if GUNSHOT>ARMR2 then PENETRATED:=true;
22220               3:
22230               if GUNSHOT>ARMR3 then PENETRATED:=true
22240               end;
22250           end;
22260           end;
22270       %*************************************************************\
22280           procedure        REPORTFIRE(II,JJ,AMM:integer);
22290           (*reports coming under fire giving the type of fire that is there *)
22300           %*************************************************************\
22310           var
22320               TYP:packed array[1..5]of char;
22330           begin
22340               if FIREREPORT[II,JJ] = true then
22350           begin
22360               case AMM of
22370               1:TYP:='SMOKE';
22380               2:TYP:='HE   ';
22390               3:TYP:='AP   ';
22400               end;
```

```
22410                 SKIP(2);FIRSREPORT[I,J]:=true;
22420                 WRITELN(TTY,'BANK',I:1,J:1,'REPORTING GUNNER RANGE',DYP,'FIRE FROM LOC'
22430                    ,FR^.
22440                    SOURCELOC.X:2,
22450                    FR^.SOURCELOC.Y:2);
22460               end;
22470               end;
22480             %*****************************************************************\
22490               procedure INITREPORTfire;
22500               (* initialises the report for tanks so that in the same burst
22510                 the fire is not reported by the same tank more than once *)
22520               %*****************************************************************\
22530               var
22540                   I,J:integer;
22550               begin
22560                   for I:=1 to 5 do
22570                   for J:=1 to 3 do
22580                     begin  FIREREPORT[I,J]:=false;
22590                   HIT[I,J]:=false;
22600                     end;
22610                   LUCHIT:=false;
22620               end;
22630             %*****************************************************************\
22640               procedure REPORTHIT(I,J:integer);
22650               (* in case the tank target has been hit this reported in this procedure *)
22660               %*****************************************************************\
22670               begin
22680                   if not HIT[I,J] then
22690               begin
22700                   TANKDATA[I,J].STATUS:=DESTRALISED:=true;
22710                   WRITELN(TTY,'BANK ',FR^.SOURCEBANK[I]:1,FR^.SOURCEBANK[2]:1,
22720                   ' REPORTS NEUTRALISING GUN',I,' IN-> LOC ',HITX:2,HITY:2);
22730                   BREAK;HIT[I,J]:=true;
22740               end;
22750               end;
22760               begin
22770             if ((I#IK)and(J#JK)) then
22780               begin
22790                   REPORTFIRE(I,J,AMMTYPE);
22800                   if ((AMMTYPE=1)or(HITYPE#1)) then
22810               begin
22820                   if DISTANCE < 1000 then DISTYPE:=0
22830                   else
22840                   if DISTANCE <2000 then DISTYPE:=1
22850                   else
22860                   if DISTANCE <4000 then DISTYPE:=2
22870                   else DISTYPE:=3;
22880                   ANGLE:=ABS(TANKDATA[IK,JK].ORIENTASN-TANKDATA[IK,JK].ORIENTASN);
22890                   if
22900                   ((ANGLE < 45)or(ANGLE >315)) then ANGLETYPE:=0
22910                   else
22920                   if ((ANGLE>135)and(ANGLE<225)) then ANGLETYPE:=1
22930                   else ANGLETYPE:=2;
22940                   if PENETRATED then
22950                   REPORTHIT(I,J)
22960                   else
```

```
22970                   begin
22980               case DISTYPE of
22990               0:
23000                   if ANGTYPE=2 then
23010                 begin
23020                   case ANGLETYPE of
23030                   1:with PNKDAT[I,JJ].STATUS do
23040                     begin
23050                       CANSHOOT:=false;
23060                       SHOCKED:=true;
23070                     end;
23080                   2:PNKDAT[I,JJ].STATUS.CANMOV:=
23090                   false;
23100                   3:REPORTHIT(I,J)
23110                   end
23120                 end
23130                   else PNKDAT[I,JJ].STATUS.SHOCKED:=true;
23140               1:
23150                   if ANGTYPE=2 then
23160                 begin
23170                   case ANGLETYPE of
23180                   1:with PNKDAT[I,JJ].STATUS do
23190                     begin
23200                       CANSHOOT:=false;
23210                       SHOCKED:=true;
23220                     end;
23230                   2:PNKDAT[I,JJ].STATUS.CANMOV:=
23240                   false;
23250                   3:REPORTHIT(I,J)
23260                   end
23270                 end
23280                   else
23290                   if ANGLETYPE=2 then PNKDAT[I,JJ].STATUS.
23300                   CANMOV:=false
23310                   else
23320                 begin
23330                   PNKDAT[I,JJ].STATUS.SHOCKED:=true;
23340                   PNKDAT[I,JJ].STATUS.CANSHOOT:=false
23350                 end;
23360               2:
23370                   if ANGTYPE=3 then PNKDAT[I,JJ].STATUS.
23380                   CANMOV:=false
23390                   else
23400                   PNKDAT[I,JJ].STATUS.SHOCKED:=true;
23410               others:
23420               end;
23430                 end;
23440           end;
23450           end;
23460           end;
23470           begin
23480       for II:=1 to 5 do
23490       for JJ:=1 to 3 do
23500         begin
23510             if ((PNKDAT[II,JJ].MAPCORD.X=HITX)and
23520             (PNKDAT[II,JJ].MAPCORD.Y= HITY)) then
```

```
23530                 ANALYISEHIT(II,JJ,0,FR^.TYPOFAMO)
23540                 else
23550                 if ((ABS(TRKDATI[II,JJ].XAPCORD.X-HITX)=1)and
23560                 (ABS(TRKDATI[II,JJ].YAPCORD.Y-HITY)= 1)) then
23570                 ANALYISEHIT(II,JJ,1,FR^.TYPOFAMO);
23580               end;
23590           if FR^.DESIRK[1]=0 then
23600           if ((HITX=FR^.DESTLOC.X)and(HITY=FR^.DESTLOC.Y)and(not LOCHIT)) then
23610             begin
23620           WRITELN(TTY,'TANK',FR^.SOURCERK[1]:1,FR^.SOURCERK[2]:1,
23630           'REPORTS GIVING EFFECTIVE FIRE ON ',HITX:4,HITY:2);
23640           LOCHIT:=true;
23650             end;
23660             end;
23670           %*****************************************************************\
23680         procedure INITREPORT(fire;
23690         (* initialises the report fire to false *)
23700         %*****************************************************************\
23710               var
23720           I,J:integer;
23730             begin
23740           for I:=1 to 5 do
23750           for J:=1 to 3 do FIREREPORT[I,J]:=false
23760             end;
23770           %*****************************************************************\
23780         procedure CALCULATERounds(FTEMP:FPTR;var ROUNDS:integer);
23790         (* calculates the rounds that are to be fired in that timestep *)
23800         %*****************************************************************\
23810               begin
23820           if FTEMP^.TIME < MINUTESTEP then
23830             begin
23840           ROUNDS:=(MINUTESTEP-FTEMP^.TIME)*FTEMP^.RATE;
23850           FTEMP^.TIME:=0;
23860           if ROUNDS > FTEMP^.POINTM then ROUNDS:=FTEMP^.POINTM
23870             end
23880           else
23890             begin
23900           FTEMP^.TIME:=FTEMP^.TIME - MINUTESTEP;
23910           ROUNDS:=0
23920             end
23930             end;
23940         begin
23950           FR:=FROOT;REWRITE(FIRED);
23960           while FR^.BLINK<>nil do
23970             begin
23980           FR:=FR^.BLINK;CALCULATERounds(FR,NO);INITREPORT(fire;LOCHIT:=false;
23990           if NO >0 then
24000             begin
24010           PLUS:=1;TRKDATI[FR^.SOURCERK[1],FR^.SOURCERK[2]].STATUS.FIRING:=true;
24020           CALCULATEDist(FR^.SOURCELOC.X,FR^.SOURCELOC.Y,FR^.DESTLOC.X,FR^.DESTLOC.
24030                   Y,DISTANCE);
24040           ASINE:=(FR^.DESTLOC.Y - FR^.SOURCELOC.Y)/DISTANCE;
24050           ACOS:=(FR^.DESTLOC.X - FR^.SOURCELOC.X)/DISTANCE;
24060           for J:=1 to NO do
24070             begin
24080               if PLUS#1 then PROB:=2* FR^.HITPROB
```

```
24090                   else PROD:=FR^.HTERROR*
24100                   TEMPDIST:=TRUNC(PROD*DISTANCE+0.5);
24110                   HITX:=FR^.SOURCEDOC.X+TRUNC(TEMPDIST*ACOS+0.5);
24120                   HITY:=FR^.SOURCEDOC.Y+TRUNC(TEMPDIST*ASIN+0.5);
24130                   WRITELN(FTREAT,HITX,HITY);
24140                   PLUS:=PLUS * (-1);
24150                   FR^.HITPROB:=SQRT(FR^.HITPROB);
24160                   CHECKHIT;
24170                 end;
24180           FR^.TOTNUM:=FR^.TOTNUM-10;
24190           case FR^.TYPEFAIL of
24200                1:TNKDAT[FR^.SOURCETK[1],FR^.SOURCETK[2]].RADCAP.TYP1:=TNKDAT[FR^.
24210                SOURCETK[1],FR^.
24220                SOURCETK[2]].RADCAP.TYP1-10;
24230                2:TNKDAT[FR^.SOURCETK[1],FR^.SOURCETK[2]].RADCAP.TYP2:=TNKDAT[FR^.
24240                SOURCETK[1],FR^.
24250                SOURCETK[2]].RADCAP.TYP2-10;
24260                3:TNKDAT[FR^.SOURCETK[1],FR^.SOURCETK[2]].RADCAP.TYP3:=TNKDAT[FR^.
24270                SOURCETK[1],FR^.
24280                SOURCETK[2]].RADCAP.TYP3-10
24290           end;
24300           if FR^.TOTNUM=0 then
24310             begin
24320                FQ:=FR;
24330                TNKDAT[FR^.SOURCETK[1],FR^.SOURCETK[2]].STATUS.FIRING:=false;
24340                FR:=FR^.ULINK;
24350                DELETEFTAB(FQ);
24360             end;
24370           end;
24380           end;
24390      end;
24400      {*****************************************************************\
24410      procedure TRANSFER(TEMP:LETTERWORD);
24420      (* transfers the words that are to be displayed on the graphics display in gnar *)
24430      {*****************************************************************\
24440      var
24450           I:integer;
24460      begin
24470           for I:=1 to 12 do
24480           NAME[I]:=TEMP[I];
24490           NAME[13]:='*';
24500           NAME[14]:='.';
24510      end;
24520           {*****************************************************************\
24530      procedure DRAWLANDMARK(LP:LMPTR);
24540      (* draws the landmarks that have been defined by the user on the graphics display *)
24550      {*****************************************************************\
24560      var
24570           X,Y:integer;
24580      begin
24590           if LP^.LLINK<>nil then DRAWLANDMARK(LP^.LLINK);
24600           if LP^.LMROOT then
24610             begin
24620           CONVERT(LP^.GR,X,Y):TRANSFER(LP^.NAME);
24630           LINE(REDUCE(X),REDUCE(Y),0);
24640           STRING(NAME);
```

```
24650           end;
24660       if BP^.RLINK<>nil then DRAWLANDMARK(BP^.RLINK);
24670   end;
24680       {*****************************************************************\
24690   procedure DRAWTANKS;
24700   (* draws the tanks at the location they are at that time, given the tank nos *)
24710   {*****************************************************************\
24720   var
24730       I,J:integer;
24740   begin
24750       TNAME[3]:='*';
24760       TNAME[4]:='.';
24770       for J:=1 to 5 do
24780         begin
24790       TNAME[1]:=CHR(ORD('0')+J);
24800       for I:=1 to 3 do
24810         begin
24820       TNAME[2]:=CHR(ORD('0')+I)
24830       ;LINE(REDUCE(TANKDAT[J,I].MAPCORD.X),REDUCE(TANKDAT[J,I].MAPCORD.Y),0);
24840       STRING(TNAME);
24850         end;
24860         end;
24870   end;
24880       {*****************************************************************\
24890   procedure DRAWFIREPT;
24900   (* draws the location of all of the spells *)
24910   {*****************************************************************\
24920   var
24930       X,Y:integer;
24940   begin while EOF(FIREDT) do
24950         begin
24960       READLN(FIREDT,X,Y);
24970       LINE(REDUCE(X),REDUCE(Y),0);
24980       MARKER(8);
24990         end
25000   end;
25010       {*****************************************************************\
25020   procedure DRAWMAP;
25030   (* draws the map on the graphic display *)
25040   {*****************************************************************\
25050   var
25060       XX,YY:real;
25070       W,SP,D,H:integer;X,Y,X1,Y1,MARK:integer;
25080       XINCR,YINCR:real;
25090       LASTX,LASTY:real;FIRST:boolean;
25100   procedure READJOBJT;(*READS DIRECTLY INTO JOBJ 12 CHARACTERS*)
25110   (* reads directly 12 char from file MAPDAT and stores them in JOBJ.
25120    used to interactive communication *)
25130   {*****************************************************************\
25140         var
25150       INCH:char;I:integer;
25160         begin
25170       for I:=1 to 12 do JOBJ[I]:=' ';
25180       INCH:=' ';I:=1;
25190       while INCH=' ' do READ(MAPDAT,INCH);
25200       while ((not EOLN(MAPDAT))and(I<13)and(INCH<>',')) do
```

```
25210             begin
25220           WORD[I]:=INCH;I:=I+1;READ(MAPDAT,INCH);
25230             end;
25240           if I>12 then MSG(5)
25250           else
25260           if EOLN(MAPDAT) then WORD[I]:=INCH;
25270           while not EOLN(MAPDAT) do READ(MAPDAT,INCH);
25280             end;
25290           %*****************************************************************\
25300         procedure DRAW;
25310         (* draws the details of the map as defined earlier *)
25320         %*****************************************************************\
25330             var
25340           LASTX,LASTY:integer;
25350           %*************************************************************\
25360             procedure FILLBETWEEN(X,LASTX,Y,LASTY:integer);
25370             (* used for get inbetween points for the display purpose *)
25380             %*************************************************************\
25390             var
25400           PX,PY:real;TEMP:integer;
25410             begin
25420           TEMP:=LASTX-X;PX:=X;PY:=Y;
25430           if TEMP=1 then PX:=X+0.5
25440           else
25450           if TEMP=(-1) then PX:=LASTX+0.5;
25460           TEMP:=LASTY-Y;
25470           if TEMP=1 then PY:=Y+0.5
25480           else
25490           if TEMP=(-1) then PY:=LASTY+0.5;
25500           if ((X#LASTX)or(Y#LASTY)) then
25510             begin
25520               LINE(PX/4),PY/4),0);MARKER(MARK);
25530             end;
25540             end;
25550             begin
25560           loop
25570           READINPUT;
25580           NUMBER;
25590           exit if not ISNUMBER;
25600           CONVERT(INTVAL,X1,Y1);
25610           LASTX:=X;LASTY:=Y;
25620           repeat
25630                 READLN(MAP,X,Y);
25640                 FILLBETWEEN(X,LASTX,Y,LASTY);
25650                 LINE(REDUCE(X),REDUCE(Y),0);
25660                 MARKER(MARK);
25670                 LASTX:=X;LASTY:=Y;
25680           until((X=X1)and(Y=Y1));
25690           end
25700             end;
25710           %*****************************************************************\
25720         procedure READCHAR;(*READS A CHARACTER FROM THE FILE MAPDAT*)
25730         %*****************************************************************\
25740             var
25750           ALP:char;
25760             begin
```

```
25770             repeat READ(MAPDAT,CH)
25780             until CH#' ';
25790             while not (EOF(MAPDAT)or(ALP='.')) do READ(MAPDAT,ALP);
25800               end;
25810         begin
25820             RESET(MAPDAT);RESET(MAP);
25830             WRITELN(TTY,'GIVE POINT OF START MAP->');BREAK;READLN(TTY);
25840             RESET(FIREDT);CLRDEV(1);
25850             LINE(0.0,0.0,0);
25860             LINE(1.0,0.0);
25870             LINE(1.0,1.0);
25880             LINE(0.0,1.0);
25890             LINE(0.0,0.0);
25900             YY:=0.0;
25910             CSIZEH(120);CSIZEV(60);
25920             for I:=1 to 7 do
25930               begin
25940             XX:=0.0;
25950             YY:=YY+0.125;
25960             for J:=1 to 7 do
25970               begin
25980             XX:=XX+0.125;
25990             LINE(XX,YY,0);
26000             MARKER(3);
26010               end;
26020               end;
26030             for I:=1 to 7 do
26040               begin
26050             loop
26060             READCHAR;
26070             exit if CH#'Y';
26080             if I<3 then
26090               begin
26100                 READLN(MAPDAT,A);READLN(MAPDAT,B);
26110               end
26120             else
26130             if I=7 then READLN(MAPDAT,N);
26140             READLN(MAPDAT,SP);
26150             CONVERT(SP,X,Y);
26160             LINE(REDUCE(X),REDUCE(Y),0);
26170             FIRST:=true;
26180             if ((I=1)or(I=3)) then
26190               begin
26200                 loop
26210                 READINPUT;NUMBER;
26220                 exit if not ISNUMBER;CONVERT(INPVAL,X1,Y1);
26230                 LINE(REDUCE(X1),REDUCE(Y1),1);XINCR:=0.0;YINCR:=0.0;
26240                 if I=1 then
26250                   begin
26260                 if X=X1 then XINCR:=-0.005
26270                 else YINCR:=0.005;
26280                 if FIRST then
26290                   begin FIRST:=false;LASTX:=REDUCE(X)+XINCR;LASTY:=
26300                 REDUCE(Y)+YINCR;
26310                   end;
26320                 LINE(LASTX,LASTY,0);
```

```
26330                 BASIX:=REDUCE(X1)+XINCR;BASIY:=REDUCE(Y1)+YINCR;
26340                 LINE(BASIX,BASIY,1);
26350                 LINE(REDUCE(X1),REDUCE(Y1),0);
26360                   end;
26370                 X:=X1;Y:=Y1;
26380                 end;
26390           end
26400         else
26410           begin
26420                 case I of
26430                 1,3:WRITELN(TTY,'ERROR');
26440                 2:MARK:=3;
26450                 4:MARK:=1;
26460                 5:MARK:=4;
26470                 6:MARK:=5;
26480                 others:MARK:=10
26490                 end;
26500                 MARKER(MARK);
26510                 DRAW;
26520           end;
26530         end
26540           end;
26550         DRAWLANDMARK(LMROOT);DRAWTANK;
26560         DRAWFIREPT;
26570         WRITELN(TTY,'GIVE <RETURN> TO CLEAR & CONTINUE');READLN(TTY);
26580         CURDEV(1);
26590       end;
26600         %*************************************************************\
26610       procedure COMNDPROS;(*PROCESSES THE TYPE OF COMMAND*)
26620       %*************************************************************\
26630       begin READWORD;
26640         if WORD='REPORT       ' then REPORTPROC
26650         else
26660         if WORD='MOVE         ' then MOVEPROCEDURE
26670         else
26680         if ((WORD='FIRE         ')or(WORD='SHOOT        ')) then FIREPROCEDURE
26690         else
26700           begin
26710         WRITELN(TTY,'YOUR STATEMENT':20,WORD:15,'NOT RECOGNISED PLEASE REVERT':30);
26720         BREAK
26730           end
26740       end;
26750         %*************************************************************\
26760       procedure TANKNAME;(*READS THE NAME OF TANK ADRESSED*)
26770       (* the no of the tank is stored in IK=tankno,JK=troopno.*)
26780       (* it then checks if the tank is nuetralised .*)
26790       (*if it is NEUTRALISED  then IT  is not allowed to be accesed *)
26800       %*************************************************************\
26810       begin
26820         READWORD;
26830         NUMBER;
26840         TANKNUMBER:=INTVAL;
26850         IK:=NUMWORD[1];JK:=NUMWORD[2];
26860         if ((IK<6)and(IK>0)and(JK>0)and(JK<4)) then
26870           begin
26880         if ((not TNKDAT[IK,JK].STATUS.NUETRALISED)and(not TNKDAT[IK,JK].STATUS.SHOCKED))
```

```
26890          then COMMDPROS;
26900            end
26910          else WRITELN(TTY,'TANK NO ',IK:1,JK:1,' WRONG GIVE AGAIN');
26920      end;
26930      %*************************************************************\
26940      procedure CHECK;
26950      (* analyises the type of CHECK required and takes actio accordingly *)
26960      %*************************************************************\
26970      begin WRITELN(TTY,'THE FOLLOWING CHECKS ARE POSSIBLE ');
26980          WRITELN(TTY,'  ':15,'LISTING LANDMARKS TYPE "L"');
26990          WRITELN(TTY,'  ':15,'LISTING CURRENTLY MOVING TANKS="T"');
27000          WRITELN(TTY,'  ':15,'DRAWINGG THE MAP "M"');
27010          WRITELN(TTY,'  ':15,'SETTING TO DRAW THE MAP="S"');
27020          WRITELN(TTY,'  ':15,'DELETING THE SET MAP DRAWING="D"');
27030          WRITELN(TTY,'  ':15,'RETURN="R"');WRITE(TTY,'CHECK REQUIRED FOR:');
27040          BREAK;READCH;
27050          case CH of
27060          'L':PRILMTAB(LMROOT);'T':PRINTMOVTAb;
27070          'M':DRAWMAP;'S':MAPREQUIREd:=true;'D':MAPREQUIREd:=false;
27080          others:
27090            begin WRITELN(TTY,'YOUR ',CH,' NOT RECOGNISED PLEASE RETYPE');
27100              CHECK
27110            end
27120          end;
27130      end;
27140      begin
27150          INITIALISE;
27160          INITGRID;
27170          INITTANK;
27180          FILLTOPOGRaphy;
27190          ENTERLM;
27200          READTIMEStep;
27210          loop
27220          SKIP(1);
27230          if not FREEZE then
27240            begin  MOVETANKS;REMOVESHOCk;FIRE;
27250            end;
27260          if MAPREQUIREd then DRAWMAP;
27270          WRITE(TTY,'YOUR COMMAND:');BREAK;
27280          UPDATETIME;
27290          READLINE;READWORD;TESTNUMBER:=false;
27300          exit if WORD='END         ';
27310          if WORD='TK          ' then
27320          TANKNAME
27330          else
27340          if (WORD='HELPCOMND   ')or (WORD='HELP        ') then HELPCOMND
27350          else
27360          if WORD='LANDMARK    ' then ENTERLM
27370          else
27380          if WORD='CHECK       ' then CHECK
27390          else
27400          if WORD='FREEZE      ' then FREEZE:=true
27410          else
27420          if ((WORD='CONTINUE    ')or(WORD='CONT        '))then FREEZE:=
27430          false
27440          else ERROR(1)
```